```
// SO2Rduino - An SO2R Box built on an Arduino clone
//
// Copyright 2010, Paul Young

// This is a simple implementation of an OTRSP SO2R Device.
// See http://www.k1xm.org/otrsp/ for information on the
// protocol.

// This implementation does not use the Arduino libraries
// for much.  It also does not use interrupts except for
// the one ms timer set up by the Arduino environment.
// As long as the main loop takes less than about 1 ms in
// the worst case it can keep up with the UART which is
// run at 9600 baud.

// Modifications made by W3SZ August 2017 to permit use with
// modern Arduino IDEs, and to correct bug where Aux2 values displayed
// with ?AUX2 command were actually AUX1's values
// further modifications by W3SZ September 2017:
// use with MEGA:
// eliminated shift register for aux outputs;
// each output (aux1 and aux2) has 16 GPIO pins for output values 0-15
// separated tx2 into cw2, mic2, ptt_a_2, ptt_b_2
// I have found this extremely helpful for vhr/uhf/microwave operation
// over the past 15 years or so.  There are now
// separate manual spdt toggles with center off position for:
// rcv, cw, mic, ptt_a, ptt_b.
// removed rx = !tx function as N1MM supplies this with dueling CQs and
// some settings of this could really mess up dueling CQs.
// I just commented it out so it can be reinstituted if desired.

#include <EEPROM.h>
#include "uart.h"

// Misc. definitions
#define DEBOUNCE 10                     // Debounce time (ms)

#define EEPROM_RESERVED 0
#define EEPROM_MONO     1
#define EEPROM_LATCH    2

// The front panel switches have three values.  They are
// declared as byte because the compiler generates better
// code that way.
typedef enum {
    RADIO_AUTO = 0,
    RADIO_1,
    RADIO_2
} radio_switch;

// Switches and inputs
static boolean          ptt_computer;   // Computer asserts PTT
static byte             ptt_debounce;   // PTT switch debounce (ms)

static byte             cw_switch;      // Front panel cw switch
static byte             mic_switch;      // Front panel mic switch
static byte             ptt_a_switch;      // Front panel ptt_a switch
static byte             ptt_b_switch;      // Front panel ptt_b switch
static byte             rx_switch;      // Front panel RX switch
static byte             cw_debounce;    // CW switch debounce (ms)
static byte             mic_debounce;    // MIC switch debounce (ms)
static byte             ptt_a_debounce;    // PTT_A switch debounce (ms)
static byte             ptt_b_debounce;    // PTT_B switch debounce (ms)
static byte             rx_debounce;    // RX switch debounce (ms)
static byte             debounce;       // Debounce timer

```

```
// Outputs
static boolean          cw2;            // Radio 2 selected for CW
static boolean          mic2;            // Radio 2 selected for MIC
static boolean          ptt_a_2;            // Radio 2 selected for PTT_A
static boolean          ptt_b_2;            // Radio 2 selected for PTT_B
static boolean          rx2;            // Listening to radio 2
static boolean          stereo;         // Listening in stereo
static boolean          tx2_computer;   // Computer says select TX 2
static boolean          rx2_computer;   // Computer says select RX 2
static boolean          stereo_computer;// Computer says RX stereo

// Aux outputs
static byte             aux1;           // Aux 1 output
static byte             aux2;           // Aux 2 output

// OTRSP events
static boolean          event_tx;       // TX event is enabled
static boolean          event_rx;       // RX event is enabled
static boolean          event_ab;       // Abort event is enabled
static boolean          event_cr0;      // PTT switch event is enabled

// Special and misc state
static boolean          mono;           // Stereo is not allowed
static boolean          latch;          // Move phones to non-TX radio

// Forward references
static boolean check_switches();
static void do_relays();
static void do_ptt();
static void do_aux();
static void check_key_ptt();
static void do_command();


//define constant pin aliases
//output pins
const int RX2 = 10; //RX2 OUTPUT (SET_RX2, CLEAR_RX2) //was D13
const int STEREO = 11; //STEREO OUTPUT (SET_STEREO, CLEAR_STEREO)
const int MIC2 = 9; //MIC2 RELAY AND LED OUTPUT (SET_TX2, CLEAR_TX2)
const int MIC1_LED = 8; //MIC2 LED OUTPUT (SET_TX1, CLEAR_TX1)
const int RX1_LED = A4; //RX1 LED OUTPUT (SET_RX1_LED, CLEAR_RX1_LED)
const int RX2_LED = A5; //RX2 LED OUTPUT (SET_RX2_LED, CLEAR_RX2_LED)
const int PTT1 = 4; //PTT 1 OUTPUT (SET_PTT1)
const int PTT2 = 5; //PTT 2 OUTPUT (SET_PTT2)
const int CW1 = 6; //CW 1 OUTPUT (SET_CW1)
const int CW2 = 7; //CW 2 OUTPUT (SET_CW2)

//input pins
const int GET_PTT_A_SW = 12; //PTT_A FOOTSWITCH INPUT (GET_PTT_SWITCH)
const int GET_PTT_B_SW = 13; //PTT_B FOOTSWITCH INPUT (GET_PTT_SWITCH)
const int GET_PTT_DTR = 2; //PTT DTR INPUT (GET_PTT_DTR) (RTS FROM PIN 14 OF CH340G)
const int GET_CW_KEY = 3; //CW KEY INPUT (GET_CW_KEY)
const int CW1_MAN = A0; //TX1 SWITCH INPUT (GET_TX1_SWITCH)
const int CW2_MAN = A1; //TX2 SWITCH INPUT (GET TX2_sWITCH)
const int RX1_MAN = A2; //RX1 SWITCH INPUT (GET_RX1_SWITCH)
const int RX2_MAN = A3; //RX2 SWITCH INPUT (GET_RX2_SWITCH)

//constants below added with conversion to MEGA with
//conversion of serial aux outputs to parallel
//both AUX1 and AUX outputs provided
//Two PTTs provided for: PTT_A and PTT_B
//each can be computer controlled or manual depending on
//setting of manual PTT_A 1-off-2 switch
//and manual PTT_B 1-off-2 switch
//GET_TX1 split into CW1_MAN and MIC1_MAN and PTT1_MAN
//GET_TX2 split into CW2_MAN and MIC2_MAN and PTT2_MAN
```

```
133: //All 3 mech switches are center off DPDT toggles
134: const int MIC1_MAN = 14;
135: const int MIC2_MAN = 15;
136: const int PTT_A_1_MAN = 16;
137: const int PTT_A_2_MAN = 17;
138: const int PTT_B_1_MAN = 18;
139: const int PTT_B_2_MAN = 19;
140:
141: //below are output pins for AUX1 and AUX2
142: const int AUX1_0 = 22;
143: const int AUX1_1 = 24;
144: const int AUX1_2 = 26;
145: const int AUX1_3 = 28;
146: const int AUX1_4 = 30;
147: const int AUX1_5 = 32;
148: const int AUX1_6 = 34;
149: const int AUX1_7 = 36;
150: const int AUX1_8 = 38;
151: const int AUX1_9 = 40;
152: const int AUX1_10 = 42;
153: const int AUX1_11 = 44;
154: const int AUX1_12 = 46;
155: const int AUX1_13 = 48;
156: const int AUX1_14 = 50;
157: const int AUX1_15 = 52;
158:
159: const int AUX2_0 = 23;
160: const int AUX2_1 = 25;
161: const int AUX2_2 = 27;
162: const int AUX2_3 = 29;
163: const int AUX2_4 = 31;
164: const int AUX2_5 = 33;
165: const int AUX2_6 = 35;
166: const int AUX2_7 = 37;
167: const int AUX2_8 = 39;
168: const int AUX2_9 = 41;
169: const int AUX2_10 = 43;
170: const int AUX2_11 = 45;
171: const int AUX2_12 = 47;
172: const int AUX2_13 = 49;
173: const int AUX2_14 = 51;
174: const int AUX2_15 = 53;
175:
176: void
177: setup()
178: //-------------------------------------------------------------------------
179: // Initialize the SO2RDuino
180: //-------------------------------------------------------------------------
181: {
182:     cw_switch = RADIO_AUTO;
183:     mic_switch = RADIO_AUTO;
184:     ptt_a_switch = RADIO_AUTO;
185:     ptt_b_switch = RADIO_AUTO;
186:     rx_switch = RADIO_AUTO;
187:     debounce = millis();
188:     rx_debounce = 0;
189:     cw_debounce = 0;
190:     mic_debounce = 0;
191:     ptt_a_debounce = 0;
192:     ptt_b_debounce = 0;
193:
194: //DEFINE GPIO PIN MODES
195: //OUTPUT pins
196: pinMode(RX2,OUTPUT);
197: pinMode(STEREO,OUTPUT);
198: pinMode(MIC2,OUTPUT);
```

```
199: pinMode(MIC1_LED,OUTPUT);
200: pinMode(RX1_LED,OUTPUT);
201: pinMode(RX2_LED,OUTPUT);
202: pinMode(PTT1,OUTPUT);
203: pinMode(PTT2,OUTPUT);
204: pinMode(CW1,OUTPUT);
205: pinMode(CW2,OUTPUT);
206:
207: //INPUT pins
208: pinMode(GET_PTT_A_SW,INPUT);
209: pinMode(GET_PTT_B_SW,INPUT);
210: pinMode(GET_PTT_DTR,INPUT);
211: pinMode(GET_CW_KEY,INPUT);
212: pinMode(CW1_MAN,INPUT);
213: pinMode(CW2_MAN,INPUT);
214: pinMode(RX1_MAN,INPUT);
215: pinMode(RX2_MAN,INPUT);
216:
217: pinMode(MIC1_MAN,INPUT);
218: pinMode(MIC2_MAN,INPUT);
219: pinMode(PTT_A_1_MAN,INPUT);
220: pinMode(PTT_A_2_MAN,INPUT);
221: pinMode(PTT_B_1_MAN,INPUT);
222: pinMode(PTT_B_2_MAN,INPUT);
223:
224: //AUXILIARY (BAND DATA) OUTPUT pins
225: pinMode(AUX1_0,OUTPUT);
226: pinMode(AUX1_1,OUTPUT);
227: pinMode(AUX1_2,OUTPUT);
228: pinMode(AUX1_3,OUTPUT);
229: pinMode(AUX1_4,OUTPUT);
230: pinMode(AUX1_5,OUTPUT);
231: pinMode(AUX1_6,OUTPUT);
232: pinMode(AUX1_7,OUTPUT);
233: pinMode(AUX1_8,OUTPUT);
234: pinMode(AUX1_9,OUTPUT);
235: pinMode(AUX1_10,OUTPUT);
236: pinMode(AUX1_11,OUTPUT);
237: pinMode(AUX1_12,OUTPUT);
238: pinMode(AUX1_13, OUTPUT);
239: pinMode(AUX1_14, OUTPUT);
240: pinMode(AUX1_15, OUTPUT);
241:
242: pinMode(AUX2_0, OUTPUT);
243: pinMode(AUX2_1, OUTPUT);
244: pinMode(AUX2_2, OUTPUT);
245: pinMode(AUX2_3, OUTPUT);
246: pinMode(AUX2_4, OUTPUT);
247: pinMode(AUX2_5, OUTPUT);
248: pinMode(AUX2_6, OUTPUT);
249: pinMode(AUX2_7, OUTPUT);
250: pinMode(AUX2_8, OUTPUT);
251: pinMode(AUX2_9, OUTPUT);
252: pinMode(AUX2_10, OUTPUT);
253: pinMode(AUX2_11, OUTPUT);
254: pinMode(AUX2_12, OUTPUT);
255: pinMode(AUX2_13, OUTPUT);
256: pinMode(AUX2_14, OUTPUT);
257: pinMode(AUX2_15, OUTPUT);
258:
259: //INITIALIZE GPIO OUTPUT PINS TO LOW
260: digitalWrite(RX2,LOW);
261: digitalWrite(STEREO,LOW);
262: digitalWrite(MIC2,LOW);
263: digitalWrite(MIC1_LED,LOW);
264: digitalWrite(RX1_LED,LOW);
```

```
265: digitalWrite(RX2_LED,LOW);
266: digitalWrite(PTT1,LOW);
267: digitalWrite(PTT2,LOW);
268: digitalWrite(CW1,LOW);
269: digitalWrite(CW2,LOW);
270:
271: digitalWrite(AUX1_0,LOW);
272: digitalWrite(AUX1_1,LOW);
273: digitalWrite(AUX1_2,LOW);
274: digitalWrite(AUX1_3,LOW);
275: digitalWrite(AUX1_4,LOW);
276: digitalWrite(AUX1_5,LOW);
277: digitalWrite(AUX1_6,LOW);
278: digitalWrite(AUX1_7,LOW);
279: digitalWrite(AUX1_8,LOW);
280: digitalWrite(AUX1_9,LOW);
281: digitalWrite(AUX1_10,LOW);
282: digitalWrite(AUX1_11,LOW);
283: digitalWrite(AUX1_12,LOW);
284: digitalWrite(AUX1_13, LOW);
285: digitalWrite(AUX1_14, LOW);
286: digitalWrite(AUX1_15, LOW);
287:
288: digitalWrite(AUX2_0, LOW);
289: digitalWrite(AUX2_1, LOW);
290: digitalWrite(AUX2_2, LOW);
291: digitalWrite(AUX2_3, LOW);
292: digitalWrite(AUX2_4, LOW);
293: digitalWrite(AUX2_5, LOW);
294: digitalWrite(AUX2_6, LOW);
295: digitalWrite(AUX2_7, LOW);
296: digitalWrite(AUX2_8, LOW);
297: digitalWrite(AUX2_9, LOW);
298: digitalWrite(AUX2_10, LOW);
299: digitalWrite(AUX2_11, LOW);
300: digitalWrite(AUX2_12, LOW);
301: digitalWrite(AUX2_13, LOW);
302: digitalWrite(AUX2_14, LOW);
303: digitalWrite(AUX2_15, LOW);
304:
305: //SET PULLUP RESISTORS ON INPUTS
306: digitalWrite(GET_PTT_A_SW,HIGH);
307: digitalWrite(GET_PTT_B_SW,HIGH);
308: digitalWrite(GET_PTT_DTR,LOW);
309: digitalWrite(GET_CW_KEY,HIGH);
310: digitalWrite(CW1_MAN,HIGH);
311: digitalWrite(CW2_MAN,HIGH);
312: digitalWrite(RX1_MAN,HIGH);
313: digitalWrite(RX2_MAN,HIGH);
314:
315: digitalWrite(MIC1_MAN,HIGH);
316: digitalWrite(MIC2_MAN,HIGH);
317: digitalWrite(PTT_A_1_MAN,HIGH);
318: digitalWrite(PTT_A_2_MAN,HIGH);
319: digitalWrite(PTT_B_1_MAN,HIGH);
320: digitalWrite(PTT_B_2_MAN,HIGH);
321:
322:
323:
324:
325:     // Initialize to TX1, RX1, mono, not transmitting
326:     cw2 = false;
327:     mic2 = false;
328:     ptt_a_2 = false;
329:     ptt_b_2 = false;
330:     rx2 = false;
```

```
331:     stereo = false;
332:     tx2_computer = false;
333:     rx2_computer = false;
334:     stereo_computer = false;
335:
336:     event_tx = false;
337:     event_rx = false;
338:     event_ab = false;
339:     event_cr0 = false;
340:
341:     aux1 = 0;
342:     aux2 = 0;
343:
344:     // The values for mono and latch are stored in EEPROM
345:     mono = (bool)EEPROM.read(EEPROM_MONO);
346:     latch = (bool)EEPROM.read(EEPROM_LATCH);
347:
348:     // Initialize the uart
349:     uart_init();
350:
351:     // Set up the aux port
352:     do_aux();
353:
354:     // Wait for the switches to settle
355:     delay(DEBOUNCE);
356:
357:     check_switches();
358:     do_relays();
359:
360: }
361:
362: void
363: loop()
364: //------------------------------------------------------------------------
365: // Main loop
366: //------------------------------------------------------------------------
367: {
368:     if (check_switches()) {
369:         do_relays();
370:     }
371:
372:     check_key_ptt();
373:
374:     if (do_uart()) {
375:         do_command();
376:         uart_clear_buffer();
377:     }
378:
379:     check_key_ptt();
380: }
381:
382: static boolean check_switches()
383: //------------------------------------------------------------------------
384: // Check front panel switches and the PTT switch
385: //------------------------------------------------------------------------
386: {
387:
388:     // When a switch opens or closes the contacts can bounce a
389:     // few times, opening and closing before it settles.  To
390:     // eliminate this the code ignores the switch for a few
391:     // milliseconds after it changes.
392:     byte now = millis();
393:     boolean changed = false;
394:
395:     // Switches are closed when LOW
396:
```

```
397:     // Only do once per millisecond max
398:     if (now == debounce) {
399:         return false;
400:     }
401:     debounce = now;
402:
403:     // Check the front-panel CW switch
404:     if (cw_debounce) {
405:         cw_debounce --;
406:     }
407:     else {
408:         if (!digitalRead(CW1_MAN)) {
409:             if (cw_switch != RADIO_1) {
410:                 cw_debounce = DEBOUNCE;
411:                 cw_switch = RADIO_1;
412:                 changed = true;
413:             }
414:         }
415:         else {
416:             if (!digitalRead(CW2_MAN)) {
417:                 if (cw_switch != RADIO_2) {
418:                     cw_debounce = DEBOUNCE;
419:                     cw_switch = RADIO_2;
420:                     changed = true;
421:                 }
422:             }
423:             else {
424:                 if (cw_switch != RADIO_AUTO) {
425:                     cw_debounce = DEBOUNCE;
426:                     cw_switch = RADIO_AUTO;
427:                     changed = true;
428:                 }
429:             }
430:         }
431:     }
432:
433:
434:     // Check the front-panel MIC switch
435:     if (mic_debounce) {
436:         mic_debounce --;
437:     }
438:     else {
439:         if (!digitalRead(MIC1_MAN)) {
440:             if (mic_switch != RADIO_1) {
441:                 mic_debounce = DEBOUNCE;
442:                 mic_switch = RADIO_1;
443:                 changed = true;
444:             }
445:         }
446:         else {
447:             if (!digitalRead(MIC2_MAN)) {
448:                 if (mic_switch != RADIO_2) {
449:                     mic_debounce = DEBOUNCE;
450:                     mic_switch = RADIO_2;
451:                     changed = true;
452:                 }
453:             }
454:             else {
455:                 if (mic_switch != RADIO_AUTO) {
456:                     mic_debounce = DEBOUNCE;
457:                     mic_switch = RADIO_AUTO;
458:                     changed = true;
459:                 }
460:             }
461:         }
462:     }
```

```
463:
464:
465:     // Check the front-panel PTT_A switch
466:     if (ptt_a_debounce) {
467:         ptt_a_debounce --;
468:     }
469:     else {
470:         if (!digitalRead(PTT_A_1_MAN)) {
471:             if (ptt_a_switch != RADIO_1) {
472:                 ptt_a_debounce = DEBOUNCE;
473:                 ptt_a_switch = RADIO_1;
474:                 changed = true;
475:             }
476:         }
477:         else {
478:             if (!digitalRead(PTT_A_2_MAN)) {
479:                 if (ptt_a_switch != RADIO_2) {
480:                     ptt_a_debounce = DEBOUNCE;
481:                     ptt_a_switch = RADIO_2;
482:                     changed = true;
483:                 }
484:             }
485:             else {
486:                 if (ptt_a_switch != RADIO_AUTO) {
487:                     ptt_a_debounce = DEBOUNCE;
488:                     ptt_a_switch = RADIO_AUTO;
489:                     changed = true;
490:                 }
491:             }
492:         }
493:     }
494:
495:
496:     // Check the front-panel PTT_B switch
497:     if (ptt_b_debounce) {
498:         ptt_b_debounce --;
499:     }
500:     else {
501:         if (!digitalRead(PTT_A_1_MAN)) {
502:             if (ptt_b_switch != RADIO_1) {
503:                 ptt_b_debounce = DEBOUNCE;
504:                 ptt_b_switch = RADIO_1;
505:                 changed = true;
506:             }
507:         }
508:         else {
509:             if (!digitalRead(PTT_A_2_MAN)) {
510:                 if (ptt_b_switch != RADIO_2) {
511:                     ptt_b_debounce = DEBOUNCE;
512:                     ptt_b_switch = RADIO_2;
513:                     changed = true;
514:                 }
515:             }
516:             else {
517:                 if (ptt_b_switch != RADIO_AUTO) {
518:                     ptt_b_debounce = DEBOUNCE;
519:                     ptt_b_switch = RADIO_AUTO;
520:                     changed = true;
521:                 }
522:             }
523:         }
524:     }
525:
526:     // Check the front-panel RX switch
527:     if (rx_debounce) {
528:         rx_debounce--;
```

```
529:     }
530:     else {
531:         if (!digitalRead(RX1_MAN)) {
532:             if (rx_switch != RADIO_1) {
533:                 rx_debounce = DEBOUNCE;
534:                 rx_switch = RADIO_1;
535:                 changed = true;
536:             }
537:         }
538:         else {
539:             if (!digitalRead(RX2_MAN)) {
540:                 if (rx_switch != RADIO_2) {
541:                     rx_debounce = DEBOUNCE;
542:                     rx_switch = RADIO_2;
543:                     changed = true;
544:                 }
545:             }
546:             else {
547:                 if (rx_switch != RADIO_AUTO) {
548:                     rx_debounce = DEBOUNCE;
549:                     rx_switch = RADIO_AUTO;
550:                     changed = true;
551:                 }
552:             }
553:         }
554:     }
555:
556:     // Check the PTT_A footswitch
557:     // Note:  PTT_A switch is LOW when active.
558:     if (ptt_a_debounce) {
559:         ptt_a_debounce--;
560:     }
561:     else {
562:         if (digitalRead(GET_PTT_A_SW)) {
563:             // Switch is open
564:             if (ptt_a_switch) {
565:                 // Switch was closed
566:                 ptt_a_debounce = DEBOUNCE;
567:                 ptt_a_switch = false;
568:                 do_ptt();
569:                 if (event_cr0) {
570:                     uart_send_prog_string(PSTR("$CR00\r"));
571:                 }
572:             }
573:         }
574:         else {
575:             // Switch is closed
576:             if (!ptt_a_switch) {
577:                 // Switch was closed
578:                 ptt_a_debounce = DEBOUNCE;
579:                 ptt_a_switch = true;
580:                 do_ptt();
581:                 if (event_cr0) {
582:                     uart_send_prog_string(PSTR("$CR01\r"));
583:                 }
584:             }
585:         }
586:     }
587:
588:     // Check the PTT_B footswitch
589:     // Note:  PTT_B switch is LOW when active.
590:     if (ptt_b_debounce) {
591:         ptt_b_debounce--;
592:     }
593:     else {
594:         if (digitalRead(GET_PTT_B_SW)) {
```

```
595:             // Switch is open
596:             if (ptt_b_switch) {
597:                 // Switch was closed
598:                 ptt_b_debounce = DEBOUNCE;
599:                 ptt_b_switch = false;
600:                 do_ptt();
601:                 if (event_cr0) {
602:                     uart_send_prog_string(PSTR("$CR00\r"));
603:                 }
604:             }
605:         }
606:         else {
607:             // Switch is closed
608:             if (!ptt_b_switch) {
609:                 // Switch was closed
610:                 ptt_b_debounce = DEBOUNCE;
611:                 ptt_b_switch = true;
612:                 do_ptt();
613:                 if (event_cr0) {
614:                     uart_send_prog_string(PSTR("$CR01\r"));
615:                 }
616:             }
617:         }
618:     }
619: 
620:     return changed;
621: }
622: 
623: static void do_ptt()
624: //---------------------------------------------------------------------
625: // Set or clear PTT
626: //---------------------------------------------------------------------
627: {
628: 
629:   //deal with PTT FROM COMPUTER
630:     if (ptt_computer) //if computer is asserting PTT
631:     {
632:       if((tx2_computer && digitalRead(PTT_A_1_MAN)) || !digitalRead(PTT_A_2_MAN))
633:           {
634:             digitalWrite(PTT2, HIGH);
635:           }
636:       if ((!tx2_computer && digitalRead(PTT_A_2_MAN)) || !digitalRead(PTT_A_1_MAN))
637:           {
638:             digitalWrite(PTT1, HIGH);
639:           }
640:     }
641:     else // if not ptt_computer
642:     {
643:       if((tx2_computer && digitalRead(PTT_A_1_MAN)) || !digitalRead(PTT_A_2_MAN) )
644:           {
645:             digitalWrite(PTT2, LOW);
646:           }
647:       if ((!tx2_computer && digitalRead(PTT_A_2_MAN)) || !digitalRead(PTT_A_1_MAN))
648:           {
649:             digitalWrite(PTT1, LOW);
650:           }
651:     }
652: 
653:     //DEAL WITH PTT FROM PTT_A
654:     if (!digitalRead(GET_PTT_A_SW)) //if PTT_A is asserting PTT
655:     {
656:       if((tx2_computer && digitalRead(PTT_A_1_MAN)) || !digitalRead(PTT_A_2_MAN))
657:           {
658:             digitalWrite(PTT2, HIGH);
659:           }
660:       if ((!tx2_computer && digitalRead(PTT_A_2_MAN)) || !digitalRead(PTT_A_1_MAN))
```

```
661:            {
662:              digitalWrite(PTT1, HIGH);
663:            }
664:     }
665:     else  // if PTT_A not asserting PTT
666:     {
667:        if((tx2_computer && digitalRead(PTT_A_1_MAN)) || !digitalRead(PTT_A_2_MAN))
668:            {
669:              digitalWrite(PTT2, LOW);
670:            }
671:        if ((!tx2_computer && digitalRead(PTT_A_2_MAN)) || !digitalRead(PTT_A_1_MAN))
672:            {
673:              digitalWrite(PTT1, LOW);
674:            }
675:     }
676:
677:    //DEAL WITH PTT FROM PTT_B
678:
679:     if (!digitalRead(GET_PTT_B_SW)) //if PTT_B is asserting PTT
680:     {
681:         if((tx2_computer && digitalRead(PTT_B_1_MAN)) || !digitalRead(PTT_B_2_MAN))
682:       {
683:              digitalWrite(PTT2, HIGH);
684:       }
685:          if ((!tx2_computer && digitalRead(PTT_B_2_MAN)) || !digitalRead(PTT_B_1_MAN))
686:            {
687:              digitalWrite(PTT1, HIGH);
688:            }
689:     }
690:     else  // if PTT_B not asserting PTT
691:     {
692:         if((tx2_computer && digitalRead(PTT_B_1_MAN)) || !digitalRead(PTT_B_2_MAN))
693:            {
694:              digitalWrite(PTT2, LOW);
695:            }
696:          if ((!tx2_computer && digitalRead(PTT_B_2_MAN)) || !digitalRead(PTT_B_1_MAN))
697:            {
698:              digitalWrite(PTT1, LOW);
699:            }
700:     }
701:
702: }
703:
704: static void do_relays()
705: //---------------------------------------------------------------------------
706: // Set the relays as appropriate
707: //---------------------------------------------------------------------------
708: {
709:     boolean tx2_current;
710:     boolean rx2_current;
711:     boolean stereo_current;
712:
713:     // The switches get priority.  Otherwise do what the computer says.
714:     if (mic_switch == RADIO_AUTO) {
715:         tx2_current = tx2_computer;
716:     }
717:     else {
718:         tx2_current = (mic_switch == RADIO_2);
719:     }
720:
721:     // RX is similar to TX except that the computer can say stereo.
722:     // The switches get priority.  Otherwise do what the computer says.
723:     if (rx_switch == RADIO_AUTO) {
724:         rx2_current = rx2_computer;
725:         stereo_current = stereo_computer && !mono;
726:     }
```

```
727:     else {
728:         rx2_current = (rx_switch == RADIO_2);
729:         stereo_current = false;
730:     }
731:
732:     if (mic2 != tx2_current) {
733:         // If the abort event is going to be sent do it before sending
734:         // the transmit changed event.  This is because it takes a few
735:         // milliseconds to send an event and the abort is more important.
736:         if (ptt_a_switch ||ptt_b_switch || ptt_computer || (!digitalRead(GET_CW_KEY)))
                 {
737:             // Transmitting while changing the transmitter.  Do what is possible
738:             // to stop this and send an abort if the event is turned on.
739:
740:         digitalWrite(CW1,LOW);
741:         digitalWrite(CW2,LOW);
742:         digitalWrite(PTT1,LOW);
743:         digitalWrite(PTT2,LOW);
744:             if (event_ab) {
745:                 uart_send_prog_string(PSTR("$AB\r"));
746:             }
747:         }
748:         if (event_tx) {
749:            uart_send_prog_string((tx2_current) ? PSTR("$TX2\r") : PSTR("$TX1\r"));
750:         }
751:     }
752:
753:     // Send a receive changed event if desired
754:     if (event_rx && ((rx2 != rx2_current) || (stereo != stereo_current))) {
755:         uart_send_prog_string((rx2_current) ? PSTR("$RX2") : PSTR("$RX1"));
756:         uart_send_prog_string((stereo_current) ? PSTR("S\r") : PSTR("\r"));
757:     }
758:
759:     mic2 = tx2_current;
760:     rx2 = rx2_current;
761:     stereo = stereo_current;
762:
763:     //below is commented out because
764:     //it is not necessary with N1MM
765:     //and in fact messes up the receiver
766:     //switching with dueling CQ
767:     /*
768:     // Do the latch after checking because it is vendor-specific and
769:     // does not cause an event.
770:     if ((rx_switch == RADIO_AUTO) && ptt_computer && latch) {
771:         rx2_current = !tx2_current;
772:         stereo_current = false;
773:     }
774:     */
775:
776:     // Now set the relays and the LEDs
777:
778:     if (tx2_current) {
779:         digitalWrite(MIC2,HIGH);
780:         digitalWrite(MIC1_LED,LOW);
781:     }
782:     else {
783:         digitalWrite(MIC1_LED,HIGH);
784:         digitalWrite(MIC2,LOW);
785:     }
786:
787:     //For stereo The RX2 relay must not be set.
788:     if (stereo_current) {
789:         digitalWrite(RX2,LOW);
790:         digitalWrite(STEREO,HIGH);
791:         digitalWrite(RX1_LED,HIGH);
```

```
792:         digitalWrite(RX2_LED,HIGH);
793:     }
794:     else {
795:         if (rx2_current) {
796:         digitalWrite(RX2,HIGH);
797:         digitalWrite(STEREO,LOW);
798:         digitalWrite(RX1_LED,LOW);
799:         digitalWrite(RX2_LED,HIGH);
800:         }
801:         else {
802:         digitalWrite(RX2,LOW);
803:         digitalWrite(STEREO,LOW);
804:         digitalWrite(RX1_LED,HIGH);
805:         digitalWrite(RX2_LED,LOW);
806:         }
807:     }
808: }
809:
810:
811: static void check_key_ptt()
812: //---------------------------------------------------------------------------
813: // Check the CW key and computer PTT
814: //---------------------------------------------------------------------------
815: {
816:     // If the key is closed the result is LOW
817:     if (digitalRead(GET_CW_KEY)) {
818:       if((tx2_computer && digitalRead(CW1_MAN)) || !digitalRead(CW2_MAN))
819:       {
820:         digitalWrite(CW2,LOW);
821:         cw2 = true;
822:       }
823:       if((!tx2_computer && digitalRead(CW2_MAN)) || !digitalRead(CW1_MAN))
824:       {
825:         digitalWrite(CW1,LOW);
826:         cw2 = false;
827:       }
828:
829:     }
830:     else {
831:       if((tx2_computer && digitalRead(CW1_MAN)) || !digitalRead(CW2_MAN))
832:       {
833:         digitalWrite(CW2,HIGH);
834:         cw2 = true;
835:       }
836:       if((!tx2_computer && digitalRead(CW2_MAN)) || !digitalRead(CW1_MAN))
837:       {
838:         digitalWrite(CW1,HIGH);
839:         cw2 = false;
840:       }
841:     }
842:     // If DTR is set the result is LOW
843:     if (digitalRead(GET_PTT_DTR)) {
844:         if (ptt_computer) {
845:             ptt_computer = false;
846:   //          do_ptt();
847:         }
848:     }
849:     else {
850:         if (!ptt_computer) {
851:             ptt_computer = true;
852:   //          do_ptt();
853:         }
854:     }
855:
856:     do_ptt();
857: }
```

```
858:
859: static void do_aux()
860: //------------------------------------------------------------------------
861: // Set the Auxiliary ports
862: // AUX1 is for Radio 1
863: // AUX2 is for Radio 2
864: //------------------------------------------------------------------------
865: {
866:     int data;
867:
868:     data = (int)aux2;
869:     switch(data)
870:     {
871:       case 0:
872:       {
873:         digitalWrite(AUX2_0, HIGH);
874:         digitalWrite(AUX2_1, LOW);
875:         digitalWrite(AUX2_2, LOW);
876:         digitalWrite(AUX2_3, LOW);
877:         digitalWrite(AUX2_4, LOW);
878:         digitalWrite(AUX2_5, LOW);
879:         digitalWrite(AUX2_6, LOW);
880:         digitalWrite(AUX2_7, LOW);
881:         digitalWrite(AUX2_8, LOW);
882:         digitalWrite(AUX2_9, LOW);
883:         digitalWrite(AUX2_10, LOW);
884:         digitalWrite(AUX2_11, LOW);
885:         digitalWrite(AUX2_12, LOW);
886:         digitalWrite(AUX2_13, LOW);
887:         digitalWrite(AUX2_14, LOW);
888:         digitalWrite(AUX2_15, LOW);
889:         break;
890:       }
891:       case 1:
892:       {
893:         digitalWrite(AUX2_0, LOW);
894:         digitalWrite(AUX2_1, HIGH);
895:         digitalWrite(AUX2_2, LOW);
896:         digitalWrite(AUX2_3, LOW);
897:         digitalWrite(AUX2_4, LOW);
898:         digitalWrite(AUX2_5, LOW);
899:         digitalWrite(AUX2_6, LOW);
900:         digitalWrite(AUX2_7, LOW);
901:         digitalWrite(AUX2_8, LOW);
902:         digitalWrite(AUX2_9, LOW);
903:         digitalWrite(AUX2_10, LOW);
904:         digitalWrite(AUX2_11, LOW);
905:         digitalWrite(AUX2_12, LOW);
906:         digitalWrite(AUX2_13, LOW);
907:         digitalWrite(AUX2_14, LOW);
908:         digitalWrite(AUX2_15, LOW);
909:         break;
910:       }
911:
912:       case 2:
913:       {
914:         digitalWrite(AUX2_0, LOW);
915:         digitalWrite(AUX2_1, LOW);
916:         digitalWrite(AUX2_2, HIGH);
917:         digitalWrite(AUX2_3, LOW);
918:         digitalWrite(AUX2_4, LOW);
919:         digitalWrite(AUX2_5, LOW);
920:         digitalWrite(AUX2_6, LOW);
921:         digitalWrite(AUX2_7, LOW);
922:         digitalWrite(AUX2_8, LOW);
923:         digitalWrite(AUX2_9, LOW);
```

```
924:          digitalWrite(AUX2_10, LOW);
925:          digitalWrite(AUX2_11, LOW);
926:          digitalWrite(AUX2_12, LOW);
927:          digitalWrite(AUX2_13, LOW);
928:          digitalWrite(AUX2_14, LOW);
929:          digitalWrite(AUX2_15, LOW);
930:          break;
931:        }
932:
933:        case 3:
934:        {
935:          digitalWrite(AUX2_0, LOW);
936:          digitalWrite(AUX2_1, LOW);
937:          digitalWrite(AUX2_2, LOW);
938:          digitalWrite(AUX2_3, HIGH);
939:          digitalWrite(AUX2_4, LOW);
940:          digitalWrite(AUX2_5, LOW);
941:          digitalWrite(AUX2_6, LOW);
942:          digitalWrite(AUX2_7, LOW);
943:          digitalWrite(AUX2_8, LOW);
944:          digitalWrite(AUX2_9, LOW);
945:          digitalWrite(AUX2_10, LOW);
946:          digitalWrite(AUX2_11, LOW);
947:          digitalWrite(AUX2_12, LOW);
948:          digitalWrite(AUX2_13, LOW);
949:          digitalWrite(AUX2_14, LOW);
950:          digitalWrite(AUX2_15, LOW);
951:          break;
952:        }
953:
954:        case 4:
955:        {
956:          digitalWrite(AUX2_0, LOW);
957:          digitalWrite(AUX2_1, LOW);
958:          digitalWrite(AUX2_2, LOW);
959:          digitalWrite(AUX2_3, LOW);
960:          digitalWrite(AUX2_4, HIGH);
961:          digitalWrite(AUX2_5, LOW);
962:          digitalWrite(AUX2_6, LOW);
963:          digitalWrite(AUX2_7, LOW);
964:          digitalWrite(AUX2_8, LOW);
965:          digitalWrite(AUX2_9, LOW);
966:          digitalWrite(AUX2_10, LOW);
967:          digitalWrite(AUX2_11, LOW);
968:          digitalWrite(AUX2_12, LOW);
969:          digitalWrite(AUX2_13, LOW);
970:          digitalWrite(AUX2_14, LOW);
971:          digitalWrite(AUX2_15, LOW);
972:          break;
973:        }
974:
975:        case 5:
976:        {
977:          digitalWrite(AUX2_0, LOW);
978:          digitalWrite(AUX2_1, LOW);
979:          digitalWrite(AUX2_2, LOW);
980:          digitalWrite(AUX2_3, LOW);
981:          digitalWrite(AUX2_4, LOW);
982:          digitalWrite(AUX2_5, HIGH);
983:          digitalWrite(AUX2_6, LOW);
984:          digitalWrite(AUX2_7, LOW);
985:          digitalWrite(AUX2_8, LOW);
986:          digitalWrite(AUX2_9, LOW);
987:          digitalWrite(AUX2_10, LOW);
988:          digitalWrite(AUX2_11, LOW);
989:          digitalWrite(AUX2_12, LOW);
```

```
 990:          digitalWrite(AUX2_13, LOW);
 991:          digitalWrite(AUX2_14, LOW);
 992:          digitalWrite(AUX2_15, LOW);
 993:          break;
 994:        }
 995:
 996:        case 6:
 997:        {
 998:          digitalWrite(AUX2_0, LOW);
 999:          digitalWrite(AUX2_1, LOW);
1000:          digitalWrite(AUX2_2, LOW);
1001:          digitalWrite(AUX2_3, LOW);
1002:          digitalWrite(AUX2_4, LOW);
1003:          digitalWrite(AUX2_5, LOW);
1004:          digitalWrite(AUX2_6, HIGH);
1005:          digitalWrite(AUX2_7, LOW);
1006:          digitalWrite(AUX2_8, LOW);
1007:          digitalWrite(AUX2_9, LOW);
1008:          digitalWrite(AUX2_10, LOW);
1009:          digitalWrite(AUX2_11, LOW);
1010:          digitalWrite(AUX2_12, LOW);
1011:          digitalWrite(AUX2_13, LOW);
1012:          digitalWrite(AUX2_14, LOW);
1013:          digitalWrite(AUX2_15, LOW);
1014:          break;
1015:        }
1016:
1017:        case 7:
1018:        {
1019:          digitalWrite(AUX2_0, LOW);
1020:          digitalWrite(AUX2_1, LOW);
1021:          digitalWrite(AUX2_2, LOW);
1022:          digitalWrite(AUX2_3, LOW);
1023:          digitalWrite(AUX2_4, LOW);
1024:          digitalWrite(AUX2_5, LOW);
1025:          digitalWrite(AUX2_6, LOW);
1026:          digitalWrite(AUX2_7, HIGH);
1027:          digitalWrite(AUX2_8, LOW);
1028:          digitalWrite(AUX2_9, LOW);
1029:          digitalWrite(AUX2_10, LOW);
1030:          digitalWrite(AUX2_11, LOW);
1031:          digitalWrite(AUX2_12, LOW);
1032:          digitalWrite(AUX2_13, LOW);
1033:          digitalWrite(AUX2_14, LOW);
1034:          digitalWrite(AUX2_15, LOW);
1035:          break;
1036:        }
1037:
1038:        case 8:
1039:        {
1040:          digitalWrite(AUX2_0, LOW);
1041:          digitalWrite(AUX2_1, LOW);
1042:          digitalWrite(AUX2_2, LOW);
1043:          digitalWrite(AUX2_3, LOW);
1044:          digitalWrite(AUX2_4, LOW);
1045:          digitalWrite(AUX2_5, LOW);
1046:          digitalWrite(AUX2_6, LOW);
1047:          digitalWrite(AUX2_7, LOW);
1048:          digitalWrite(AUX2_8, HIGH);
1049:          digitalWrite(AUX2_9, LOW);
1050:          digitalWrite(AUX2_10, LOW);
1051:          digitalWrite(AUX2_11, LOW);
1052:          digitalWrite(AUX2_12, LOW);
1053:          digitalWrite(AUX2_13, LOW);
1054:          digitalWrite(AUX2_14, LOW);
1055:          digitalWrite(AUX2_15, LOW);
```

```
1056:         break;
1057:       }
1058:
1059:       case 9:
1060:       {
1061:         digitalWrite(AUX2_0, LOW);
1062:         digitalWrite(AUX2_1, LOW);
1063:         digitalWrite(AUX2_2, LOW);
1064:         digitalWrite(AUX2_3, LOW);
1065:         digitalWrite(AUX2_4, LOW);
1066:         digitalWrite(AUX2_5, LOW);
1067:         digitalWrite(AUX2_6, LOW);
1068:         digitalWrite(AUX2_7, LOW);
1069:         digitalWrite(AUX2_8, LOW);
1070:         digitalWrite(AUX2_9, HIGH);
1071:         digitalWrite(AUX2_10, LOW);
1072:         digitalWrite(AUX2_11, LOW);
1073:         digitalWrite(AUX2_12, LOW);
1074:         digitalWrite(AUX2_13, LOW);
1075:         digitalWrite(AUX2_14, LOW);
1076:         digitalWrite(AUX2_15, LOW);
1077:         break;
1078:       }
1079:
1080:       case 10:
1081:       {
1082:         digitalWrite(AUX2_0, LOW);
1083:         digitalWrite(AUX2_1, LOW);
1084:         digitalWrite(AUX2_2, LOW);
1085:         digitalWrite(AUX2_3, LOW);
1086:         digitalWrite(AUX2_4, LOW);
1087:         digitalWrite(AUX2_5, LOW);
1088:         digitalWrite(AUX2_6, LOW);
1089:         digitalWrite(AUX2_7, LOW);
1090:         digitalWrite(AUX2_8, LOW);
1091:         digitalWrite(AUX2_9, LOW);
1092:         digitalWrite(AUX2_10, HIGH);
1093:         digitalWrite(AUX2_11, LOW);
1094:         digitalWrite(AUX2_12, LOW);
1095:         digitalWrite(AUX2_13, LOW);
1096:         digitalWrite(AUX2_14, LOW);
1097:         digitalWrite(AUX2_15, LOW);
1098:         break;
1099:       }
1100:
1101:       case 11:
1102:       {
1103:         digitalWrite(AUX2_0, LOW);
1104:         digitalWrite(AUX2_1, LOW);
1105:         digitalWrite(AUX2_2, LOW);
1106:         digitalWrite(AUX2_3, LOW);
1107:         digitalWrite(AUX2_4, LOW);
1108:         digitalWrite(AUX2_5, LOW);
1109:         digitalWrite(AUX2_6, LOW);
1110:         digitalWrite(AUX2_7, LOW);
1111:         digitalWrite(AUX2_8, LOW);
1112:         digitalWrite(AUX2_9, LOW);
1113:         digitalWrite(AUX2_10, LOW);
1114:         digitalWrite(AUX2_11, HIGH);
1115:         digitalWrite(AUX2_12, LOW);
1116:         digitalWrite(AUX2_13, LOW);
1117:         digitalWrite(AUX2_14, LOW);
1118:         digitalWrite(AUX2_15, LOW);
1119:         break;
1120:       }
1121:
```

```
1122:       case 12:
1123:       {
1124:         digitalWrite(AUX2_0, LOW);
1125:         digitalWrite(AUX2_1, LOW);
1126:         digitalWrite(AUX2_2, LOW);
1127:         digitalWrite(AUX2_3, LOW);
1128:         digitalWrite(AUX2_4, LOW);
1129:         digitalWrite(AUX2_5, LOW);
1130:         digitalWrite(AUX2_6, LOW);
1131:         digitalWrite(AUX2_7, LOW);
1132:         digitalWrite(AUX2_8, LOW);
1133:         digitalWrite(AUX2_9, LOW);
1134:         digitalWrite(AUX2_10, LOW);
1135:         digitalWrite(AUX2_11, LOW);
1136:         digitalWrite(AUX2_12, HIGH);
1137:         digitalWrite(AUX2_13, LOW);
1138:         digitalWrite(AUX2_14, LOW);
1139:         digitalWrite(AUX2_15, LOW);
1140:         break;
1141:       }
1142:
1143:       case 13:
1144:       {
1145:         digitalWrite(AUX2_0, LOW);
1146:         digitalWrite(AUX2_1, LOW);
1147:         digitalWrite(AUX2_2, LOW);
1148:         digitalWrite(AUX2_3, LOW);
1149:         digitalWrite(AUX2_4, LOW);
1150:         digitalWrite(AUX2_5, LOW);
1151:         digitalWrite(AUX2_6, LOW);
1152:         digitalWrite(AUX2_7, LOW);
1153:         digitalWrite(AUX2_8, LOW);
1154:         digitalWrite(AUX2_9, LOW);
1155:         digitalWrite(AUX2_10, LOW);
1156:         digitalWrite(AUX2_11, LOW);
1157:         digitalWrite(AUX2_12, LOW);
1158:         digitalWrite(AUX2_13, HIGH);
1159:         digitalWrite(AUX2_14, LOW);
1160:         digitalWrite(AUX2_15, LOW);
1161:         break;
1162:       }
1163:
1164:       case 14:
1165:       {
1166:         digitalWrite(AUX2_0, LOW);
1167:         digitalWrite(AUX2_1, LOW);
1168:         digitalWrite(AUX2_2, LOW);
1169:         digitalWrite(AUX2_3, LOW);
1170:         digitalWrite(AUX2_4, LOW);
1171:         digitalWrite(AUX2_5, LOW);
1172:         digitalWrite(AUX2_6, LOW);
1173:         digitalWrite(AUX2_7, LOW);
1174:         digitalWrite(AUX2_8, LOW);
1175:         digitalWrite(AUX2_9, LOW);
1176:         digitalWrite(AUX2_10, LOW);
1177:         digitalWrite(AUX2_11, LOW);
1178:         digitalWrite(AUX2_12, LOW);
1179:         digitalWrite(AUX2_13, LOW);
1180:         digitalWrite(AUX2_14, HIGH);
1181:         digitalWrite(AUX2_15, LOW);
1182:         break;
1183:       }
1184:
1185:       case 15:
1186:       {
1187:         digitalWrite(AUX2_0, LOW);
```

```
1188:         digitalWrite(AUX2_1, LOW);
1189:         digitalWrite(AUX2_2, LOW);
1190:         digitalWrite(AUX2_3, LOW);
1191:         digitalWrite(AUX2_4, LOW);
1192:         digitalWrite(AUX2_5, LOW);
1193:         digitalWrite(AUX2_6, LOW);
1194:         digitalWrite(AUX2_7, LOW);
1195:         digitalWrite(AUX2_8, LOW);
1196:         digitalWrite(AUX2_9, LOW);
1197:         digitalWrite(AUX2_10, LOW);
1198:         digitalWrite(AUX2_11, LOW);
1199:         digitalWrite(AUX2_12, LOW);
1200:         digitalWrite(AUX2_13, LOW);
1201:         digitalWrite(AUX2_14, LOW);
1202:         digitalWrite(AUX2_15, HIGH);
1203:         break;
1204:       }
1205:     }
1206:
1207:
1208:     data = (int)aux1;
1209:     switch(data)
1210:     {
1211:       case 0:
1212:       {
1213:         digitalWrite(AUX1_0, HIGH);
1214:         digitalWrite(AUX1_1, LOW);
1215:         digitalWrite(AUX1_2, LOW);
1216:         digitalWrite(AUX1_3, LOW);
1217:         digitalWrite(AUX1_4, LOW);
1218:         digitalWrite(AUX1_5, LOW);
1219:         digitalWrite(AUX1_6, LOW);
1220:         digitalWrite(AUX1_7, LOW);
1221:         digitalWrite(AUX1_8, LOW);
1222:         digitalWrite(AUX1_9, LOW);
1223:         digitalWrite(AUX1_10, LOW);
1224:         digitalWrite(AUX1_11, LOW);
1225:         digitalWrite(AUX1_12, LOW);
1226:         digitalWrite(AUX1_13, LOW);
1227:         digitalWrite(AUX1_14, LOW);
1228:         digitalWrite(AUX1_15, LOW);
1229:         break;
1230:       }
1231:       case 1:
1232:       {
1233:         digitalWrite(AUX1_0, LOW);
1234:         digitalWrite(AUX1_1, HIGH);
1235:         digitalWrite(AUX1_2, LOW);
1236:         digitalWrite(AUX1_3, LOW);
1237:         digitalWrite(AUX1_4, LOW);
1238:         digitalWrite(AUX1_5, LOW);
1239:         digitalWrite(AUX1_6, LOW);
1240:         digitalWrite(AUX1_7, LOW);
1241:         digitalWrite(AUX1_8, LOW);
1242:         digitalWrite(AUX1_9, LOW);
1243:         digitalWrite(AUX1_10, LOW);
1244:         digitalWrite(AUX1_11, LOW);
1245:         digitalWrite(AUX1_12, LOW);
1246:         digitalWrite(AUX1_13, LOW);
1247:         digitalWrite(AUX1_14, LOW);
1248:         digitalWrite(AUX1_15, LOW);
1249:         break;
1250:       }
1251:
1252:       case 2:
1253:       {
```

```
1254:         digitalWrite(AUX1_0, LOW);
1255:         digitalWrite(AUX1_1, LOW);
1256:         digitalWrite(AUX1_2, HIGH);
1257:         digitalWrite(AUX1_3, LOW);
1258:         digitalWrite(AUX1_4, LOW);
1259:         digitalWrite(AUX1_5, LOW);
1260:         digitalWrite(AUX1_6, LOW);
1261:         digitalWrite(AUX1_7, LOW);
1262:         digitalWrite(AUX1_8, LOW);
1263:         digitalWrite(AUX1_9, LOW);
1264:         digitalWrite(AUX1_10, LOW);
1265:         digitalWrite(AUX1_11, LOW);
1266:         digitalWrite(AUX1_12, LOW);
1267:         digitalWrite(AUX1_13, LOW);
1268:         digitalWrite(AUX1_14, LOW);
1269:         digitalWrite(AUX1_15, LOW);
1270:         break;
1271:       }
1272:
1273:       case 3:
1274:       {
1275:         digitalWrite(AUX1_0, LOW);
1276:         digitalWrite(AUX1_1, LOW);
1277:         digitalWrite(AUX1_2, LOW);
1278:         digitalWrite(AUX1_3, HIGH);
1279:         digitalWrite(AUX1_4, LOW);
1280:         digitalWrite(AUX1_5, LOW);
1281:         digitalWrite(AUX1_6, LOW);
1282:         digitalWrite(AUX1_7, LOW);
1283:         digitalWrite(AUX1_8, LOW);
1284:         digitalWrite(AUX1_9, LOW);
1285:         digitalWrite(AUX1_10, LOW);
1286:         digitalWrite(AUX1_11, LOW);
1287:         digitalWrite(AUX1_12, LOW);
1288:         digitalWrite(AUX1_13, LOW);
1289:         digitalWrite(AUX1_14, LOW);
1290:         digitalWrite(AUX1_15, LOW);
1291:         break;
1292:       }
1293:
1294:       case 4:
1295:       {
1296:         digitalWrite(AUX1_0, LOW);
1297:         digitalWrite(AUX1_1, LOW);
1298:         digitalWrite(AUX1_2, LOW);
1299:         digitalWrite(AUX1_3, LOW);
1300:         digitalWrite(AUX1_4, HIGH);
1301:         digitalWrite(AUX1_5, LOW);
1302:         digitalWrite(AUX1_6, LOW);
1303:         digitalWrite(AUX1_7, LOW);
1304:         digitalWrite(AUX1_8, LOW);
1305:         digitalWrite(AUX1_9, LOW);
1306:         digitalWrite(AUX1_10, LOW);
1307:         digitalWrite(AUX1_11, LOW);
1308:         digitalWrite(AUX1_12, LOW);
1309:         digitalWrite(AUX1_13, LOW);
1310:         digitalWrite(AUX1_14, LOW);
1311:         digitalWrite(AUX1_15, LOW);
1312:         break;
1313:       }
1314:
1315:       case 5:
1316:       {
1317:         digitalWrite(AUX1_0, LOW);
1318:         digitalWrite(AUX1_1, LOW);
1319:         digitalWrite(AUX1_2, LOW);
```

```
1320:           digitalWrite(AUX1_3, LOW);
1321:           digitalWrite(AUX1_4, LOW);
1322:           digitalWrite(AUX1_5, HIGH);
1323:           digitalWrite(AUX1_6, LOW);
1324:           digitalWrite(AUX1_7, LOW);
1325:           digitalWrite(AUX1_8, LOW);
1326:           digitalWrite(AUX1_9, LOW);
1327:           digitalWrite(AUX1_10, LOW);
1328:           digitalWrite(AUX1_11, LOW);
1329:           digitalWrite(AUX1_12, LOW);
1330:           digitalWrite(AUX1_13, LOW);
1331:           digitalWrite(AUX1_14, LOW);
1332:           digitalWrite(AUX1_15, LOW);
1333:           break;
1334:         }
1335:
1336:         case 6:
1337:         {
1338:           digitalWrite(AUX1_0, LOW);
1339:           digitalWrite(AUX1_1, LOW);
1340:           digitalWrite(AUX1_2, LOW);
1341:           digitalWrite(AUX1_3, LOW);
1342:           digitalWrite(AUX1_4, LOW);
1343:           digitalWrite(AUX1_5, LOW);
1344:           digitalWrite(AUX1_6, HIGH);
1345:           digitalWrite(AUX1_7, LOW);
1346:           digitalWrite(AUX1_8, LOW);
1347:           digitalWrite(AUX1_9, LOW);
1348:           digitalWrite(AUX1_10, LOW);
1349:           digitalWrite(AUX1_11, LOW);
1350:           digitalWrite(AUX1_12, LOW);
1351:           digitalWrite(AUX1_13, LOW);
1352:           digitalWrite(AUX1_14, LOW);
1353:           digitalWrite(AUX1_15, LOW);
1354:           break;
1355:         }
1356:
1357:         case 7:
1358:         {
1359:           digitalWrite(AUX1_0, LOW);
1360:           digitalWrite(AUX1_1, LOW);
1361:           digitalWrite(AUX1_2, LOW);
1362:           digitalWrite(AUX1_3, LOW);
1363:           digitalWrite(AUX1_4, LOW);
1364:           digitalWrite(AUX1_5, LOW);
1365:           digitalWrite(AUX1_6, LOW);
1366:           digitalWrite(AUX1_7, HIGH);
1367:           digitalWrite(AUX1_8, LOW);
1368:           digitalWrite(AUX1_9, LOW);
1369:           digitalWrite(AUX1_10, LOW);
1370:           digitalWrite(AUX1_11, LOW);
1371:           digitalWrite(AUX1_12, LOW);
1372:           digitalWrite(AUX1_13, LOW);
1373:           digitalWrite(AUX1_14, LOW);
1374:           digitalWrite(AUX1_15, LOW);
1375:           break;
1376:         }
1377:
1378:         case 8:
1379:         {
1380:           digitalWrite(AUX1_0, LOW);
1381:           digitalWrite(AUX1_1, LOW);
1382:           digitalWrite(AUX1_2, LOW);
1383:           digitalWrite(AUX1_3, LOW);
1384:           digitalWrite(AUX1_4, LOW);
1385:           digitalWrite(AUX1_5, LOW);
```

```
1386:         digitalWrite(AUX1_6, LOW);
1387:         digitalWrite(AUX1_7, LOW);
1388:         digitalWrite(AUX1_8, HIGH);
1389:         digitalWrite(AUX1_9, LOW);
1390:         digitalWrite(AUX1_10, LOW);
1391:         digitalWrite(AUX1_11, LOW);
1392:         digitalWrite(AUX1_12, LOW);
1393:         digitalWrite(AUX1_13, LOW);
1394:         digitalWrite(AUX1_14, LOW);
1395:         digitalWrite(AUX1_15, LOW);
1396:         break;
1397:       }
1398:
1399:       case 9:
1400:       {
1401:         digitalWrite(AUX1_0, LOW);
1402:         digitalWrite(AUX1_1, LOW);
1403:         digitalWrite(AUX1_2, LOW);
1404:         digitalWrite(AUX1_3, LOW);
1405:         digitalWrite(AUX1_4, LOW);
1406:         digitalWrite(AUX1_5, LOW);
1407:         digitalWrite(AUX1_6, LOW);
1408:         digitalWrite(AUX1_7, LOW);
1409:         digitalWrite(AUX1_8, LOW);
1410:         digitalWrite(AUX1_9, HIGH);
1411:         digitalWrite(AUX1_10, LOW);
1412:         digitalWrite(AUX1_11, LOW);
1413:         digitalWrite(AUX1_12, LOW);
1414:         digitalWrite(AUX1_13, LOW);
1415:         digitalWrite(AUX1_14, LOW);
1416:         digitalWrite(AUX1_15, LOW);
1417:         break;
1418:       }
1419:
1420:       case 10:
1421:       {
1422:         digitalWrite(AUX1_0, LOW);
1423:         digitalWrite(AUX1_1, LOW);
1424:         digitalWrite(AUX1_2, LOW);
1425:         digitalWrite(AUX1_3, LOW);
1426:         digitalWrite(AUX1_4, LOW);
1427:         digitalWrite(AUX1_5, LOW);
1428:         digitalWrite(AUX1_6, LOW);
1429:         digitalWrite(AUX1_7, LOW);
1430:         digitalWrite(AUX1_8, LOW);
1431:         digitalWrite(AUX1_9, LOW);
1432:         digitalWrite(AUX1_10, HIGH);
1433:         digitalWrite(AUX1_11, LOW);
1434:         digitalWrite(AUX1_12, LOW);
1435:         digitalWrite(AUX1_13, LOW);
1436:         digitalWrite(AUX1_14, LOW);
1437:         digitalWrite(AUX1_15, LOW);
1438:         break;
1439:       }
1440:
1441:       case 11:
1442:       {
1443:         digitalWrite(AUX1_0, LOW);
1444:         digitalWrite(AUX1_1, LOW);
1445:         digitalWrite(AUX1_2, LOW);
1446:         digitalWrite(AUX1_3, LOW);
1447:         digitalWrite(AUX1_4, LOW);
1448:         digitalWrite(AUX1_5, LOW);
1449:         digitalWrite(AUX1_6, LOW);
1450:         digitalWrite(AUX1_7, LOW);
1451:         digitalWrite(AUX1_8, LOW);
```

```
1452:         digitalWrite(AUX1_9, LOW);
1453:         digitalWrite(AUX1_10, LOW);
1454:         digitalWrite(AUX1_11, HIGH);
1455:         digitalWrite(AUX1_12, LOW);
1456:         digitalWrite(AUX1_13, LOW);
1457:         digitalWrite(AUX1_14, LOW);
1458:         digitalWrite(AUX1_15, LOW);
1459:         break;
1460:       }
1461:
1462:       case 12:
1463:       {
1464:         digitalWrite(AUX1_0, LOW);
1465:         digitalWrite(AUX1_1, LOW);
1466:         digitalWrite(AUX1_2, LOW);
1467:         digitalWrite(AUX1_3, LOW);
1468:         digitalWrite(AUX1_4, LOW);
1469:         digitalWrite(AUX1_5, LOW);
1470:         digitalWrite(AUX1_6, LOW);
1471:         digitalWrite(AUX1_7, LOW);
1472:         digitalWrite(AUX1_8, LOW);
1473:         digitalWrite(AUX1_9, LOW);
1474:         digitalWrite(AUX1_10, LOW);
1475:         digitalWrite(AUX1_11, LOW);
1476:         digitalWrite(AUX1_12, HIGH);
1477:         digitalWrite(AUX1_13, LOW);
1478:         digitalWrite(AUX1_14, LOW);
1479:         digitalWrite(AUX1_15, LOW);
1480:         break;
1481:       }
1482:
1483:       case 13:
1484:       {
1485:         digitalWrite(AUX1_0, LOW);
1486:         digitalWrite(AUX1_1, LOW);
1487:         digitalWrite(AUX1_2, LOW);
1488:         digitalWrite(AUX1_3, LOW);
1489:         digitalWrite(AUX1_4, LOW);
1490:         digitalWrite(AUX1_5, LOW);
1491:         digitalWrite(AUX1_6, LOW);
1492:         digitalWrite(AUX1_7, LOW);
1493:         digitalWrite(AUX1_8, LOW);
1494:         digitalWrite(AUX1_9, LOW);
1495:         digitalWrite(AUX1_10, LOW);
1496:         digitalWrite(AUX1_11, LOW);
1497:         digitalWrite(AUX1_12, LOW);
1498:         digitalWrite(AUX1_13, HIGH);
1499:         digitalWrite(AUX1_14, LOW);
1500:         digitalWrite(AUX1_15, LOW);
1501:         break;
1502:       }
1503:
1504:       case 14:
1505:       {
1506:         digitalWrite(AUX1_0, LOW);
1507:         digitalWrite(AUX1_1, LOW);
1508:         digitalWrite(AUX1_2, LOW);
1509:         digitalWrite(AUX1_3, LOW);
1510:         digitalWrite(AUX1_4, LOW);
1511:         digitalWrite(AUX1_5, LOW);
1512:         digitalWrite(AUX1_6, LOW);
1513:         digitalWrite(AUX1_7, LOW);
1514:         digitalWrite(AUX1_8, LOW);
1515:         digitalWrite(AUX1_9, LOW);
1516:         digitalWrite(AUX1_10, LOW);
1517:         digitalWrite(AUX1_11, LOW);
```

```
1518:         digitalWrite(AUX1_12, LOW);
1519:         digitalWrite(AUX1_13, LOW);
1520:         digitalWrite(AUX1_14, HIGH);
1521:         digitalWrite(AUX1_15, LOW);
1522:         break;
1523:       }
1524:
1525:       case 15:
1526:       {
1527:         digitalWrite(AUX1_0, LOW);
1528:         digitalWrite(AUX1_1, LOW);
1529:         digitalWrite(AUX1_2, LOW);
1530:         digitalWrite(AUX1_3, LOW);
1531:         digitalWrite(AUX1_4, LOW);
1532:         digitalWrite(AUX1_5, LOW);
1533:         digitalWrite(AUX1_6, LOW);
1534:         digitalWrite(AUX1_7, LOW);
1535:         digitalWrite(AUX1_8, LOW);
1536:         digitalWrite(AUX1_9, LOW);
1537:         digitalWrite(AUX1_10, LOW);
1538:         digitalWrite(AUX1_11, LOW);
1539:         digitalWrite(AUX1_12, LOW);
1540:         digitalWrite(AUX1_13, LOW);
1541:         digitalWrite(AUX1_14, LOW);
1542:         digitalWrite(AUX1_15, HIGH);
1543:         break;
1544:       }
1545:     }
1546: }
1547:
1548:
1549: static void do_command()
1550: //--------------------------------------------------------------------------
1551: // Parse and handle a command from the computer
1552: //--------------------------------------------------------------------------
1553: {
1554:     // Commands are not checked very thoroughly - the computer
1555:     // should not send garbage.
1556:
1557: #define COMPARE(command)  (memcmp_P(in_buf, PSTR(command), \
1558:                                     sizeof(command)-1) == 0)
1559:
1560: #define QCOMPARE(command) (memcmp_P(in_buf+1, PSTR(command), \
1561:                                     sizeof(command)-1) == 0)
1562:
1563: #define AFTER(command) (sizeof(command)-1)
1564:
1565: // Parse the commands
1566:
1567:     // Handle the queries
1568:     if (in_buf[0] == '?') {
1569:
1570:         // Check for 'ping' - just the ? by itself
1571:         if (in_buf[AFTER("?")] == '\0') {
1572:             uart_send_prog_string(PSTR("?\r"));
1573:             return;
1574:         }
1575:
1576:         // Return TX1 or TX2
1577:         if (QCOMPARE("TX")) {
1578:             uart_send_prog_string((mic2) ? PSTR("TX2\r") : PSTR("TX1\r"));
1579:            return;
1580:         }
1581:
1582:         // Return RX1 or RX2 or RX1S or RX2S
1583:         if (QCOMPARE("RX")) {
```

```
1584:             uart_send_prog_string((rx2) ? PSTR("RX2") : PSTR("RX1"));
1585:             uart_send_prog_string((stereo) ? PSTR("S\r") : PSTR("\r"));
1586:             return;
1587:         }
1588:
1589:         // Return AUX1 value
1590:         if (QCOMPARE("AUX1")) {
1591:             uart_send_prog_string(PSTR("AUX1"));
1592:             if (aux1 >= 10) {
1593:                 uart_send_char('1');
1594:                 uart_send_char((aux1-10) + '0');
1595:             }
1596:             else {
1597:                 uart_send_char(aux1 + '0');
1598:             }
1599:             uart_send_char('\r');
1600:             return;
1601:         }
1602:
1603:         // Return AUX2 value
1604:         if (QCOMPARE("AUX2")) {
1605:             uart_send_prog_string(PSTR("AUX2"));
1606:             if (aux2 >= 10) {
1607:                 uart_send_char('1');
1608:                 uart_send_char((aux2-10) + '0');
1609:             }
1610:             else {
1611:                 uart_send_char(aux2 + '0');
1612:                 uart_send_char('\r');
1613:              }
1614:             return;
1615:         }
1616:
1617:         // Return the SO2R device's name
1618:         if (QCOMPARE("NAME")) {
1619:             uart_send_prog_string(PSTR("NAME_SUPER_MEGA_SO2Rduino\r"));
1620:             return;
1621:         }
1622:
1623:         // Return the state of the footswitch
1624:         if (QCOMPARE("CR0")) {
1625:             uart_send_prog_string((ptt_a_switch || ptt_b_switch)
1626:                 ? PSTR("CR01\r") : PSTR("CR00\r"));
1627:             return;
1628:         }
1629:
1630:         // Return whether footswitch events are enabled
1631:         if (QCOMPARE("ECR0")) {
1632:             uart_send_prog_string((event_cr0)
1633:                 ? PSTR("ECR01\r") : PSTR("ECR00\r"));
1634:             return;
1635:         }
1636:
1637:         // Return whether receiver events are enabled
1638:         if (COMPARE("ERX")) {
1639:             uart_send_prog_string((event_rx)
1640:                 ? PSTR("ERX1\r") : PSTR("ERX0\r"));
1641:             return;
1642:         }
1643:
1644:         // Return whether transmitter events are enabled
1645:         if (COMPARE("ETX")) {
1646:             uart_send_prog_string((event_tx)
1647:                 ? PSTR("ETX1\r") : PSTR("ETX0\r"));
1648:             return;
1649:         }
```

```
1650:
1651:         // Return whether mono is forced (no stereo allowed)
1652:         if (COMPARE("VMONO")) {
1653:             uart_send_prog_string((mono)
1654:                 ? PSTR("VMONO1\r") : PSTR("VMONO0\r"));
1655:             return;
1656:             }
1657:
1658:         // Return whether latch feature is on
1659:         if (COMPARE("VLATCH")) {
1660:             uart_send_prog_string((latch)
1661:                 ? PSTR("VLATCH1\r") : PSTR("VLATCH0\r"));
1662:             return;
1663:         }
1664:
1665:         // Unknown query command
1666:         uart_send_string(in_buf);
1667:         uart_send_char('\r');
1668:         return;
1669:     }
1670:
1671:     // Handle transmitter change
1672:     if (COMPARE("TX")) {
1673:         tx2_computer = (in_buf[AFTER("TX")] == '2');
1674:         do_relays();
1675:        return;
1676:     }
1677:
1678:     // Handle receiver change
1679:     if (COMPARE("RX")) {
1680:         rx2_computer = (in_buf[AFTER("TX")] == '2');
1681:         stereo_computer = ((in_buf[AFTER("TXn")] == 'S') ||
1682:                   (in_buf[AFTER("TXn")] == 'R'));
1683:         do_relays();
1684:         return;
1685:     }
1686:
1687:     // Handle aux output 1 - values are four bits
1688:     if (COMPARE("AUX1")) {
1689:         aux1 = atoi((char *)&in_buf[AFTER("AUXn")]) & 15;
1690:         do_aux();
1691:         return;
1692:     }
1693:
1694:     // Handle aux output 2 - values are four bits
1695:     if (COMPARE("AUX2")) {
1696:         aux2 = atoi((char *)&in_buf[AFTER("AUXn")]) & 15;
1697:         do_aux();
1698:         return;
1699:     }
1700:
1701:     // Turn on or off PTT footswitch events
1702:     if (COMPARE("ECR0")) {
1703:       event_cr0 = (in_buf[AFTER("ECRn")] == '1');
1704:         if (event_cr0) {
1705:             uart_send_prog_string((ptt_a_switch || ptt_b_switch)
1706:                 ? PSTR("$CR01\r") : PSTR("$CR00\r"));
1707:         }
1708:         return;
1709:     }
1710:
1711:     // Turn on or off receiver events
1712:     if (COMPARE("ERX")) {
1713:         event_rx = (in_buf[AFTER("ERX")] == '1');
1714:         if (event_rx) {
1715:             uart_send_prog_string((rx2) ? PSTR("$RX2") : PSTR("$RX1"));
```

```
1716:             uart_send_prog_string((stereo) ? PSTR("S\r") : PSTR("\r"));
1717:         }
1718:         return;
1719:     }
1720:
1721:     // Turn on or off transmitter events
1722:     if (COMPARE("ETX")) {
1723:         event_tx = (in_buf[AFTER("ETX")] == '1');
1724:         if (event_tx) {
1725:              uart_send_prog_string((mic2) ? PSTR("$TX2\r") : PSTR("$TX1\r"));
1726:         }
1727:         return;
1728:     }
1729:
1730:     // Turn on or off forced mono feature
1731:     if (COMPARE("VMONO")) {
1732:         boolean n;
1733:         n = (in_buf[AFTER("VMONO")] == '1');
1734:         if (n != mono) {
1735:             mono = n;
1736:             EEPROM.write(EEPROM_MONO, mono);
1737:             do_relays();
1738:         }
1739:         return;
1740:     }
1741:
1742:     // Turn on or off latch feature
1743:     if (COMPARE("VLATCH")) {
1744:         boolean n;
1745:         n = (in_buf[AFTER("VLATCH")] == '1');
1746:         if (n != latch) {
1747:             latch = n;
1748:             EEPROM.write(EEPROM_LATCH, latch);
1749:             do_relays();
1750:         }
1751:         return;
1752:     }
1753:
1754:     // Unknown commands are ignored, as are commands that try to
1755:     // set something which is read-only such as NAME or CR0.
1756:     return;
1757: }
```

```
  1: // SO2Rduino - An SO2R Box built on an Arduino clone
  2: //
  3: // Copyright 2010, Paul Young
  4: //
  5: // This file contains the UART routines
  6:
  7: #include "Arduino.h"
  8: #include "avr/pgmspace.h"
  9: #include "uart.h"
 10:
 11: char                    in_buf[UBLEN];  // UART input buffer
 12: static byte             in_len;         // Number of chars in buffer
 13: static char             out_buf[UBLEN]; // UART output circular buffer
 14: static byte             out_add;        // Place to add a character
 15: static byte             out_remove;     // Place to remove a character
 16:
 17: void
 18: uart_send_char(const char c)
 19: //------------------------------------------------------------------------
 20: // Add a character to the UART output buffer
 21: //------------------------------------------------------------------------
 22: {
 23:
 24:     // Check for space in the buffer
 25:     if ((out_remove == (out_add + 1)) ||
 26:         ((out_remove == 0) && out_add == (UBLEN-1))) {
 27:             return;
 28:     }
 29:
 30:     out_buf[out_add++] = c;
 31:     if (out_add == UBLEN) {
 32:         out_add = 0;
 33:     }
 34: }
 35:
 36: void
 37: uart_send_string(const char* bp)
 38: //------------------------------------------------------------------------
 39: // Add a character string to the UART output buffer
 40: //------------------------------------------------------------------------
 41: {
 42:     while (*bp) {
 43:         uart_send_char(*bp++);
 44:     }
 45: }
 46:
 47: void
 48: uart_send_prog_string(const char * bp)
 49: //------------------------------------------------------------------------
 50: // Add a character string from flash to the UART output buffer
 51: //------------------------------------------------------------------------
 52: {
 53:     byte c;
 54:     while ((c=pgm_read_byte(bp++))) {
 55:         uart_send_char(c);
 56:     }
 57: }
 58:
 59: void
 60: uart_init()
 61: //------------------------------------------------------------------------
 62: // Set up the UART
 63: //------------------------------------------------------------------------
 64: {
 65:
 66:     in_len = 0;
```

```
 67:     out_add = 0;
 68:     out_remove = 0;
 69:
 70:     // Set the UART to 9600 baud
 71: #define MYUBRR 103
 72:
 73:     UBRR0H = (MYUBRR>>8);
 74:     UBRR0L = (unsigned char)MYUBRR;
 75:
 76:     // Enable receiver and transmitter
 77:     UCSR0B = _BV(RXEN0)| _BV(TXEN0);
 78:
 79:     // Set 8 bit, one stop bit
 80:     UCSR0C = _BV(UCSZ00) | _BV(UCSZ01);
 81: }
 82:
 83: void
 84: uart_clear_buffer()
 85: //---------------------------------------------------------------------------
 86: // Clear the UART input buffer
 87: //---------------------------------------------------------------------------
 88: {
 89:     in_len = 0;
 90: }
 91:
 92: boolean
 93: do_uart()
 94: //---------------------------------------------------------------------------
 95: // Send and receive characters
 96: //---------------------------------------------------------------------------
 97: {
 98:     // Check for characters to send and the UART being ready
 99:     if ((out_add != out_remove) && (UCSR0A & _BV(UDRE0))) {
100:         UDR0 = out_buf[out_remove++];
101:         if (out_remove == UBLEN) {
102:             out_remove = 0;
103:         }
104:     }
105:
106:     // Check for UART having a character to input
107:     if (UCSR0A & _BV(RXC0)) {
108:         char c = UDR0;
109:         // Special handling for return, means end of command
110:         if (c == '\r') {
111:             in_buf[in_len] = '\0';
112:             return true;
113:         }
114:         if (in_len < (UBLEN-2)) {
115:             in_buf[in_len++] = c;
116:         }
117:     }
118:
119:     return false;
120: }
121:
```

```
// SO2Rduino - An SO2R Box built on an Arduino clone
//
// Copyright 2010, Paul Young
//
// This file contains the UART prototypes
//

#ifndef UART_H

// The uart file is C, not C++ so this is needed to cause the
// compiler to generate the correct routine names.
#ifdef __cplusplus
extern "C"{
#endif

// The input buffer is used in command parsing.
#define UBLEN 64                        // UART Buffer length
extern char in_buf[UBLEN];  // UART input buffer

extern void uart_send_char(const char c);
extern void uart_send_string(const char* bp);
extern void uart_send_prog_string(const char * bp);
extern void uart_init(void);
extern void uart_clear_buffer(void);
extern boolean do_uart(void);

#ifdef __cplusplus
}
#endif

#endif

```