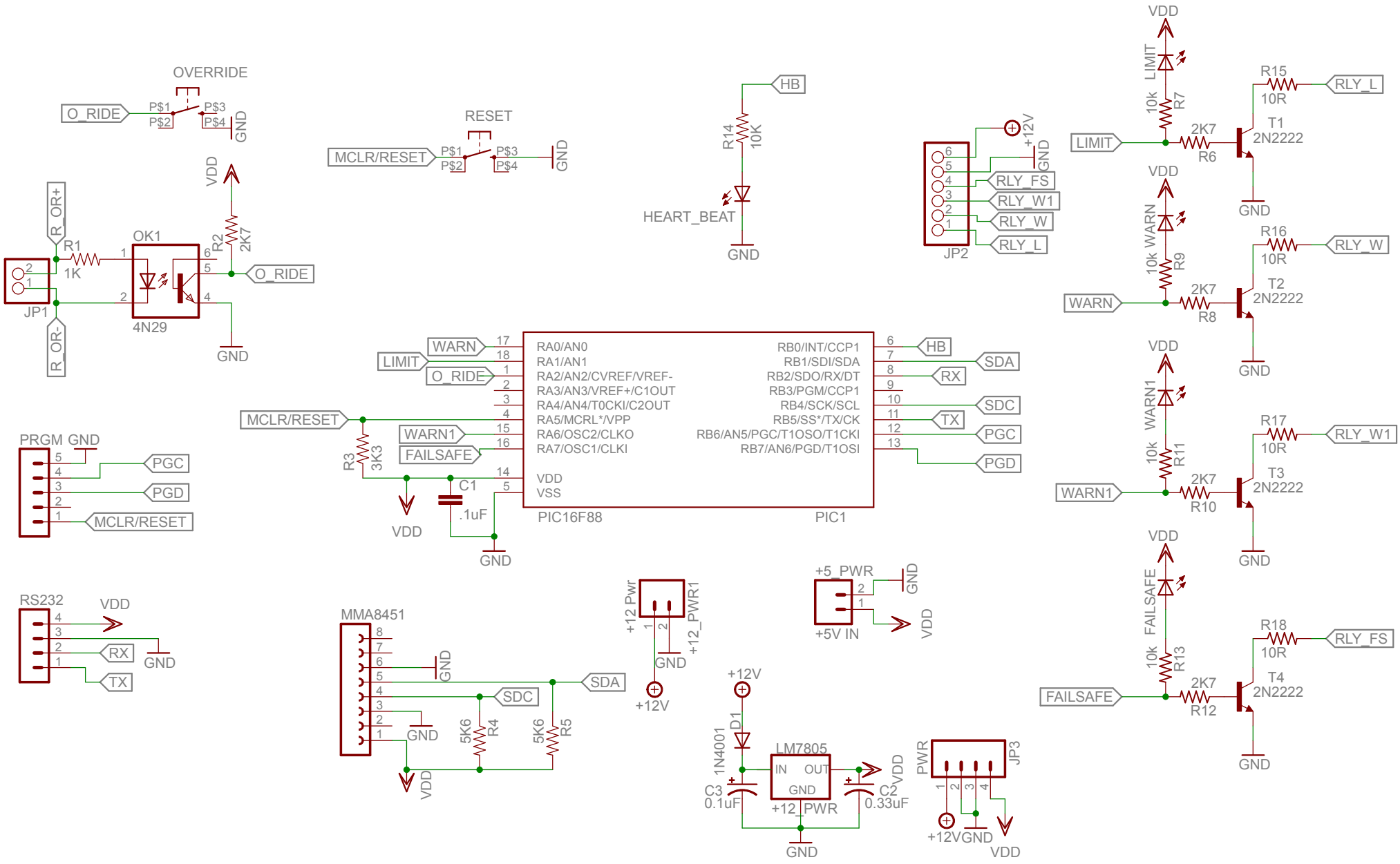
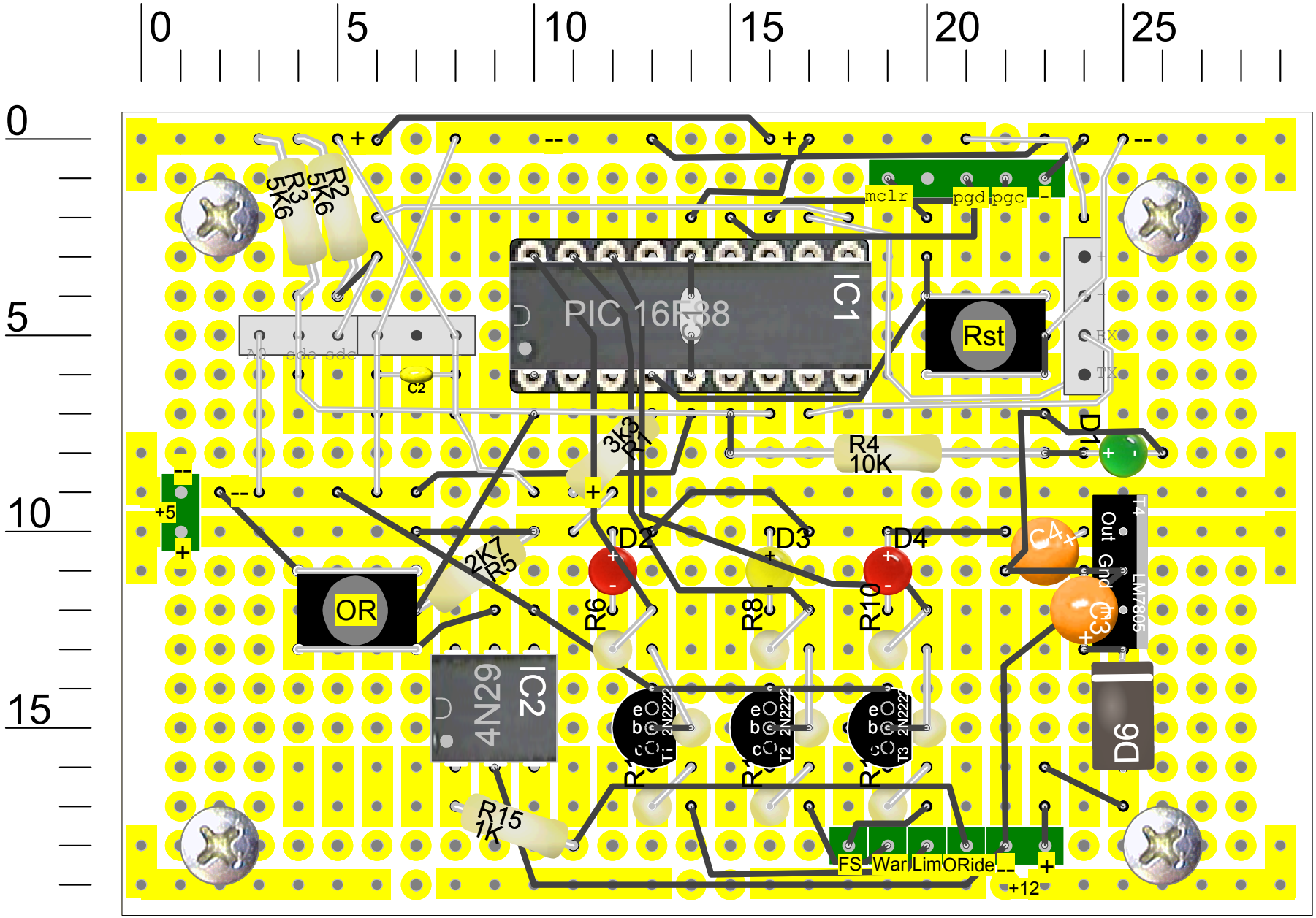
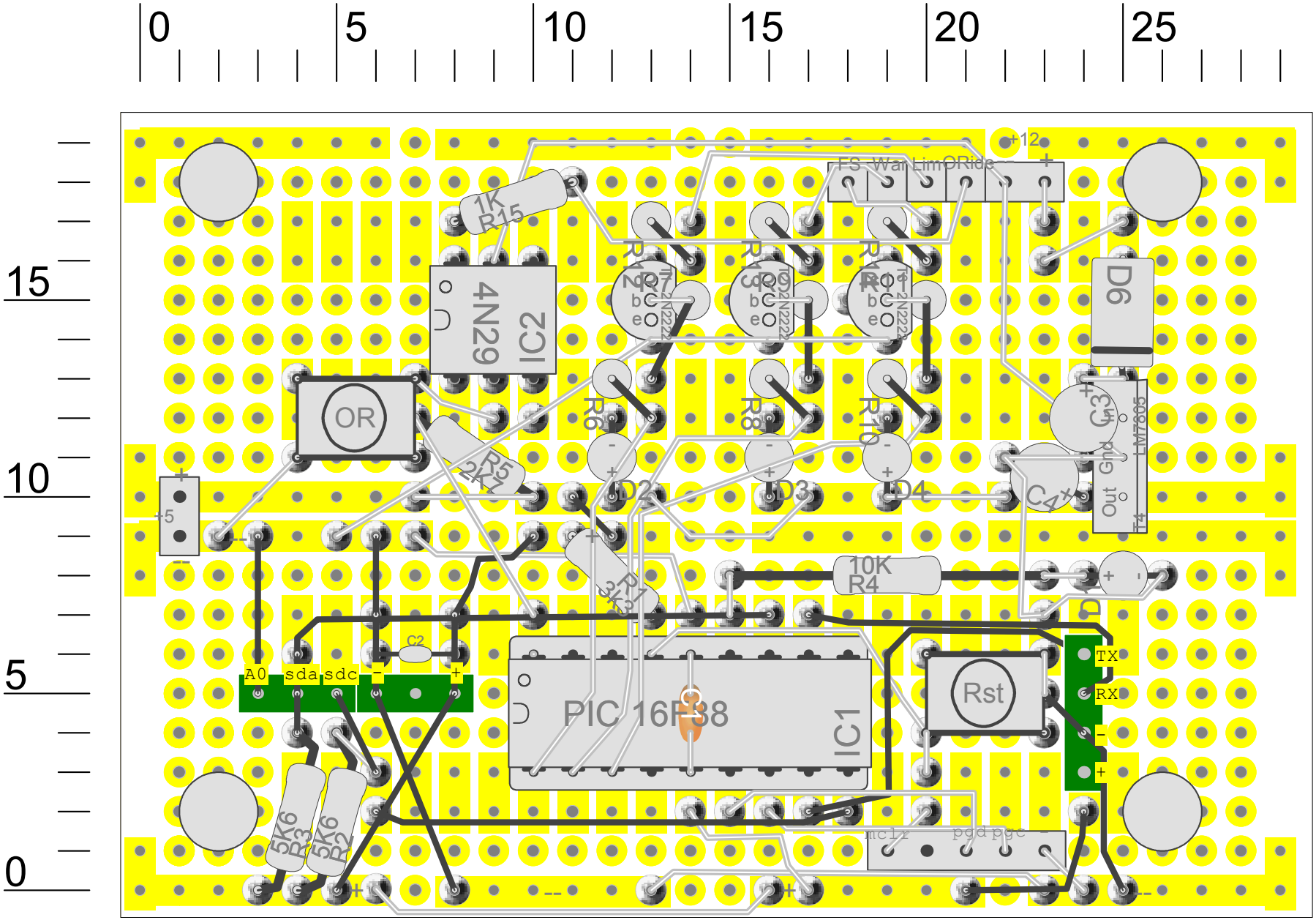
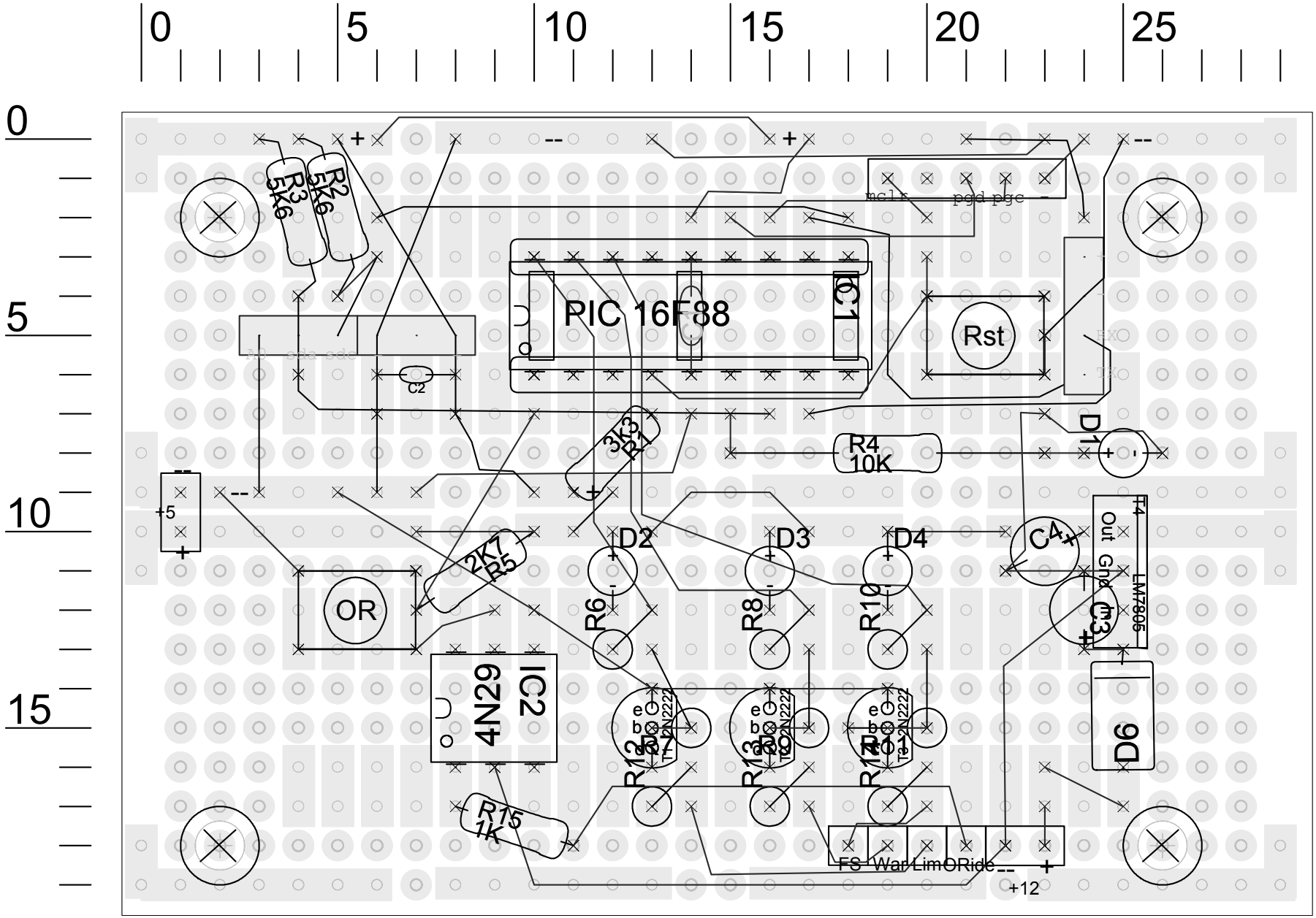
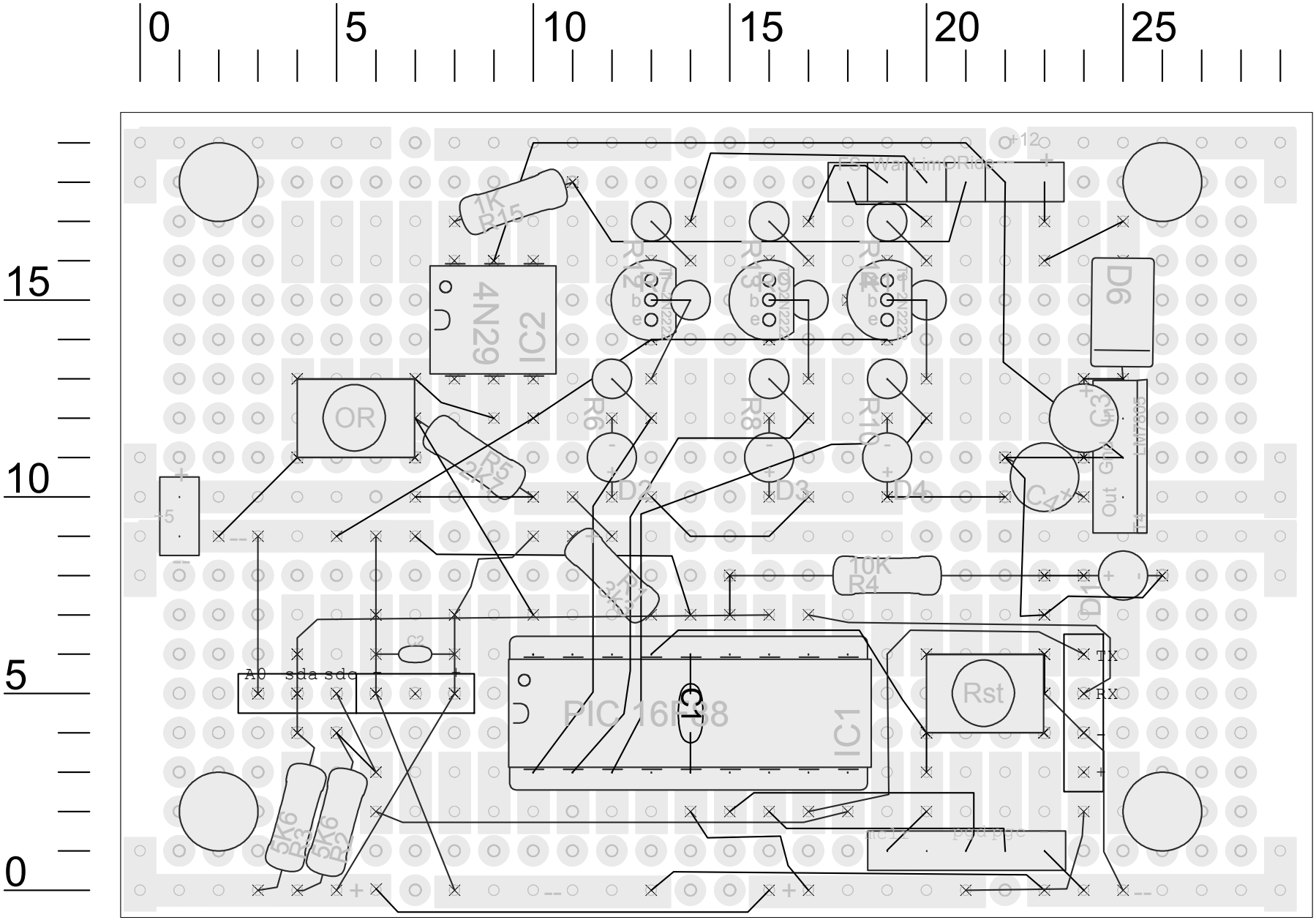

OVERRIDE
O_RIDE
P$1
P$2
P$3
P$4
GND
RESET
MCLR/RESET
HB
R14
10K
HEART_BEAT
+12V
JP2
RLY_FS
RLY_W1
RLY_W
RLY_L
VDD
LIMIT
10k
R7
R15
10R
RLY_L
T1
2N2222
2K7
R6
WARN
R9
R16
R8
T2
WARN1
R11
R17
R10
T3
FAILSAFE
R13
R18
R12
T4
R_OR+
R_OR-
R1
1K
OK1
4N29
R2
2K7
JP1
PRGM
PGC
PGD
MCLR/RESET
R3
3K3
C1
.1uF
WARN
LIMIT
O_RIDE
WARN1
FAILSAFE
RA0/AN0
RA1/AN1
RA2/AN2/CVREF/VREF-
RA3/AN3/VREF+/C1OUT
RA4/AN4/T0CKI/C2OUT
RA5/MCRL*/VPP
RA6/OSC2/CLKO
RA7/OSC1/CLKI
VDD
VSS
RB0/INT/CCP1
RB1/SDI/SDA
RB2/SDO/RX/DT
RB3/PGM/CCP1
RB4/SCK/SCL
RB5/SS*/TX/CK
RB6/AN5/PGC/T1OSO/T1CKI
RB7/AN6/PGD/T1OSI
HB
SDA
RX
SDC
TX
PGC
PGD
PIC16F88
PIC1
RS232
TX
RX
MMA8451
SDC
SDA
5K6
R4
R5
+12 Pwr
+12_PWR1
+5_PWR
+5V IN
D1
1N4001
LM7805
IN
OUT
C3
0.1uF
C2
0.33uF
+12_PWR
PWR
JP3


0
5
10
15
20
25
R2 5K6
R3 5K6
PIC 16F88
IC1
Rst
mclr
pgd
pgc
C2
R1 3K3
R4 10K
D1
+5
OR
R5 2K7
D2
D3
D4
R6
R8
R10
C4
C3
LM7805
Out
Gnd
IC2
4N29
T1 2N2222
T2 2N2222
T3 2N2222
D6
R15 1K
FS
War
Lim
ORide
+12


0
5
10
15
20
25
0
5
10
15
+12
FS
War
Lim
ORide
1K
R15
4N29
IC2
OR
2N2222
R7
R8
R9
R6
R8
R10
D6
C3
LM7805
Out
Gnd
In
C4
D2
D3
D4
+5
R5
2K2
10K
R4
R7
3K3
C2
A0
sda
sdc
PIC 16F88
IC1
Rst
TX
RX
5K6
R3
5K6
R2
mclr
pgd
pgc


0
5
10
15
20
25
PIC 16F88
IC1
Rst
mclr
pgd pgc
R2 5K6
R3 5K6
C2
R1 3K3
R4 10K
D1
+5
2K7 R5
OR
D2
D3
D4
R6
R8
R10
C4
C3
LM7805
Out Gnd
IC2
4N29
R12
R13
R14
R15 1K
TIP2N2222
D6
FS WarLimORide
+12


0
5
10
15
20
25
15
10
5
0
+12
1K
R15
4N29
IC2
OR
R5
330
R6
R7
R8
R9
R10
D2
D3
D4
D6
C3
LM7805
Out
Gnd
C4
+5
10K
R4
C2
A0
sda
sdc
PIC16F88
IC1
C1
Rst
TX
RX
5K6
R3
5K6
R2

```
 1: #define Lo(param) ((char *)&param)[0]
 2: #define Hi(param) ((char *)&param)[1]
 3:
 4: #define MMA8451_ADDR 0x38
 5: #define true 1
 6: #define false 0
 7: #define WARNINGZ 2061
 8: #define LIMITZ 1806
 9: #define WARING2Z 1870
10: #define FAILSAFEZ 1543
11:
12: #define ON 0
13: #define OFF 1
14:
15: // eprom R/W Leds
16: sbit LIMIT at RA1_bit;   //Red Led
17: sbit LIMIT_DIR at TRISA1_bit;  //Needs to be set to 0 for output
18: sbit WARNING at RA0_bit;   //Orange Led
19: sbit WARNING_DIR at TRISA0_bit;  //Needs to be set to 0 for output
20: sbit WARNING2 at RA6_bit;   //Orange Led
21: sbit WARNING2_DIR at TRISA6_bit;  //Needs to be set to 0 for output
22: sbit OVERRIDE_BTN at RA2_bit;   //Orange Led
23: sbit OVERRIDE_BTN_DIR at TRISA2_bit;  //Needs to be set to 1 for output
24: sbit FAILSAFE at RA7_bit;   //Orange Led
25: sbit FAILSAFE_DIR at TRISA7_bit;  //Needs to be set to 0 for output
26: sbit HEARTBEAT at RB0_bit;   //Green Led
27: sbit HEARTBEAT_DIR at TRISB0_bit;  //Needs to be set to 0 for output
28:
29: char MMA8451_error;
30:
31: //Software I2C Globals
32: sbit Soft_I2C_Scl           at RB4_bit;
33: sbit Soft_I2C_Sda           at RB1_bit;
34: sbit Soft_I2C_Scl_Direction at TRISB4_bit;
35: sbit Soft_I2C_Sda_Direction at TRISB1_bit;
36:
37: unsigned char MMA8451_start(unsigned short MMA8451_Addr, unsigned char mem_addr);
38: char MMA8451_read(unsigned short MMA8451_Addr, unsigned char mem_addr);
39: char MMA8451_readblk(unsigned short MMA8451_Addr, unsigned int mem_addr, char *buf
    ffer,unsigned short bsize);
40: char MMA8451_write(unsigned short MMA8451_Addr, unsigned int mem_addr, char edata
    );
41: void activate();
42: void testMode();
43: unsigned char selfTest();
44: void selfTestFail();
45: void read_xyz(unsigned char hb);
46: void output_xyz();
47: void enterOverride();
48:
49: char txt[60];
50: unsigned char accelbuf[7];
51: int xaccel=0,yaccel=0,zaccel=0;
52: long int sx,sy,sz ;
53: unsigned short heartbeatCounter=0;
54: int lHyster=0,wHyster=0;
55: #define HYSTER 20
56:
57: void test1() {
58: unsigned short i;
59: char value;
```

```
60:       for (i=0; i<16;i++) {
61:         ShortToStr(i,txt);
62:         uart1_write_text(txt);
63:         uart1_write_text(": ");
64:         value=MMA8451_read( MMA8451_ADDR, i);
65: /*Soft_I2C_Start();
66:         Soft_I2C_Write(0x38);
67:         Soft_I2C_Write(0x00);  //start at address 0
68:         Soft_I2C_Start();
69:         Soft_I2C_Write(0x38+1);
70:         value=Soft_I2C_Read(0);      // Read  byte
71:         Soft_I2C_Stop();             // Issue stop signal*/
72:         byteToHex(value,txt);
73:         txt[2]=10;
74:         txt[3]=13;
75:         txt[4]=0;
76:         uart1_write_text(txt);
77:       }
78: }
79: void test2() {
80:       unsigned short i;
81:       int value;
82:
83:       MMA8451_readblk(MMA8451_ADDR,0x00,accelbuf,7);
84:       for (i=2; i<7;i=i+2) {
85:         ShortToStr(i,txt);
86:         uart1_write_text(txt);
87:         uart1_write_text(": ");
88:         byteToHex(accelbuf[i],txt);
89:         uart1_write_text(txt);
90:         Lo(value)=accelbuf[i];
91:         Hi(value)=accelbuf[i-1];
92:         value=value/4;
93:         intTostr(value,txt);
94:         uart1_write_text(": ");
95:         uart1_write_text(txt);
96:         uart1_write_text("\r\n");
97:       }
98:
99: }
100: void test3() {
101:       unsigned short i;
102:       int value;
103:
104:       strcpy(&txt[0],"S: ");
105:       MMA8451_readblk(MMA8451_ADDR,0x00,accelbuf,7);
106:       byteToHex( accelbuf[0],&txt[strlen(txt)]);
107:
108:       strcat(txt,", X: ");
109:       Hi(value)=accelbuf[1];Lo(value)=accelbuf[2];
110:       intToStr( value/4,&txt[strlen(txt)]);
111:
112:       strcat(txt,", Y: ");
113:       Hi(value)=accelbuf[3];Lo(value)=accelbuf[4];
114:       intToStr(value/4,&txt[strlen(txt)]);
115:
116:       strcat(txt,", Z: ");
117:       Hi(value)=accelbuf[5];Lo(value)=accelbuf[6];
118:       intToStr(value/4,&txt[strlen(txt)]);
119:       uart1_write_text(txt);
120:       uart1_write_text("\r\n");
121: }
```

```
122: void test4() {
123: #define MEAN 5L
124:      unsigned short i;
125:      long int value;
126: /*x=(long int) xaccel;
127:      y=(long int) yaccel;
128:      z=(long int) zaccel;*/
129:      read_xyz(1);
130:      sx=(MEAN-1L)*((sx + ((long int) xaccel)*100L)/MEAN);
131:      xaccel=(int) (sx/(MEAN-1L)/100L);
132:      sy=(MEAN-1L)*((sy + ((long int) yaccel)*100L)/MEAN);
133:      yaccel=(int) (sy/(MEAN-1L)/100L);
134:      sz=(MEAN-1L)*((sz + ((long int) zaccel)*100L)/MEAN);
135:      zaccel=(int) (sz/(MEAN-1L)/100L);
136:      strcpy(&txt[0],"S: ");
137:      byteToHex( accelbuf[0],&txt[strlen(txt)]);
138:
139:      strcat(txt,", X: ");
140:      intToStr( xaccel,&txt[strlen(txt)]);
141:
142:      strcat(txt,", Y: ");
143:      intToStr(yaccel,&txt[strlen(txt)]);
144:
145:      strcat(txt,", Z: ");
146:      intToStr(zaccel,&txt[strlen(txt)]);
147:
148:      uart1_write_text(txt);
149:      uart1_write_text("\r\n");
150: }
151: void main() {
152: unsigned short i,st_count;
153: unsigned char override_state=0;
154: char value;
155:
156:   OSCCON=0x7c;   //Osc -> 8MHz
157:     //Set TRISx bits to 1 for input, 0 for output
158:   TRISB = 0x00;    //All output
159:   TRISA = 0x04;    //RA2 is the Override button
160:   ANSEL = 0x00;    //No analogue
161: //Clear Limits
162:   LIMIT=OFF;
163:   WARNING=OFF;
164:   WARNING2=OFF;
165:   FAILSAFE=OFF;
166:   HEARTBEAT=1;
167:   PSA_bit=0;
168:   WDTCON=0b00010110;
169:   asm CLRWDT;
170:   SWDTEN_bit=1;  //Enable WDT
171:
172:
173:
174:   UART1_Init(9600);//Serial potr setup
175:   Soft_I2C_Init(); //I2C Init
176:
177:   vdelay_ms(500);
178:
179:   activate(); //Configure the MMA8451
180:   read_xyz(1); //Get some initial readings
181:
182: /*//Initialize running mean
183:   sx = ((long int) xaccel)*(MEAN-1L)*100L;
```

```
184:   sy = ((long int) yaccel)*(MEAN-1L)*100L;
185:   sz = ((long int) zaccel)*(MEAN-1L)*100L;*/
186:
187:   override_state=0;
188:   st_count=0;
189:   //Run the self test pass==0, fail==1
190:   if (selfTest()==1) SelfTestFail();
191:   //Start looping forever
192:   do {
193:      //test4();
194:      asm CLRWDT;
195: /*if (st_count++ > 64) {
196:         selfTest();
197:         st_count=0;
198:      }*/
199:      read_xyz(1);
200:      if (zaccel < FAILSAFEZ+lHyster) {
201:         FAILSAFE=ON;
202:         LIMIT=ON;
203:         WARNING=OFF;
204:         lHyster=HYSTER;
205:      }else if (zaccel < LIMITZ+lHyster) {
206:         LIMIT=ON;
207:         WARNING=OFF;
208:         lHyster=HYSTER;
209:         if (OVERRIDE_BTN==ON) {  //has to be pressed for 3 reads
210:
211:            if (override_state<=3) { //button needs to be unpressed to activate ove
    eeride again
212:               override_state++;
213:               if (override_state==3) enterOverride();
214:            }
215:         } else {
216:            override_state=0; //reset state to unpressed
217:         }
218:      } else {
219:         if (zaccel < WARNINGZ+wHyster) {
220:            WARNING=ON;
221:            LIMIT=OFF;
222:            lHyster=0;
223:            wHyster=HYSTER;
224:         } else {
225:            WARNING=OFF;
226:            LIMIT=OFF;
227:            lHyster=0;
228:            wHyster=0;
229:         }
230:      }
231:      output_xyz();
232:   } while (true)   ;
233: }
234:
235: ////////////////////////////////////////////////////////
236: void selfTestFail() {
237:   while (true) {
238:         LIMIT = ON;
239:         WARNING = ON;
240:         HEARTBEAT = ON;
241:         asm CLRWDT;
242:         vdelay_ms( 100);
243:         HEARTBEAT =OFF;
244:         vdelay_ms( 100);
```

```
245:   }
246:   return ;
247:
248: }
249: ////////////////////////////////////////////////////////
250: void enterOverride() {
251:      int orZlimit;  //Holds current Altitude less 20 counts
252:      int loopCount; //Counts to 1 minute then exits override mode
253:      loopCount=0;
254:      orZlimit=zaccel-20;  //If zaccel gets smaller than this then cancel the overr
     ride
255:      LIMIT=OFF;   //cancel the limit
256:      WARNING=ON;  //Make it a WARNING
257:      do {
258:        asm CLRWDT;
259:        read_xyz(1);
260:        if (zaccel<orZlimit) {
261:           LIMIT=ON;   //Reinstate the LIMIT
262:           WARNING=OFF;
263:           return;
264:        }
265:        if (zaccel > LIMITZ+HYSTER) { //Cancel Overide we're safe again
266:           WARNING=OFF;
267:           return;
268:        }
269:        if (orZlimit<zaccel-25) orZlimit=zaccel-25;
270:
271:        loopCount++;
272:        output_xyz();
273:      } while (loopCount<384); //exit after 1 minute (60 * 6.4Hz))
274: }
275: void read_xyz(unsigned char hb) {
276:
277:   while ( (MMA8451_read(MMA8451_ADDR,0x00)&0x0f) != 0x0f ) {vdelay_ms(1);}  //wait
     t for new x,y,z
278:   if (heartbeatCounter++ >=10) {HEARTBEAT=1;heartbeatCounter=0;}
279:   MMA8451_readblk(MMA8451_ADDR,0x00,accelbuf,7);
280:   if (hb) HEARTBEAT=0;
281:   Hi(xaccel)=accelbuf[1];Lo(xaccel)=accelbuf[2];
282:   xaccel=xaccel/4;
283:   Hi(yaccel)=accelbuf[3];Lo(yaccel)=accelbuf[4];
284:   yaccel=yaccel/4;
285:   Hi(zaccel)=accelbuf[5];Lo(zaccel)=accelbuf[6];
286:   zaccel=zaccel/4;
287: }
288:
289: ////////////////////////////////////////////////////////
290: void activate() {
291:   char value;
292:   MMA8451_write(MMA8451_ADDR,0x2A,0x00);  //standby
293:   value=MMA8451_read(MMA8451_ADDR,0x2B);
294:   value=(value & 0b01100111) | 0b00010000 ;     //Set high res mode bits, clear te
     st-mode
295:   MMA8451_write(MMA8451_ADDR,0x2b,value);       //write new ctrl_reg2
296:   MMA8451_write(MMA8451_ADDR,0x0e,0x00);        //Range +/- 2g
297:   //MMA8451_write(MMA8451_ADDR,0x31,0x00);        //Cal_Z offset
298:   MMA8451_write(MMA8451_ADDR,0x2a,0b10110101);  //6.25hz,low noise, + goto active
      mode
299: }
300: ////////////////////////////////////////////////////////
301: void testMode(){
302:   char value;
```

```
303:   MMA8451_write(MMA8451_ADDR,0x2A,0x00);  //goto standby mode
304:   value=MMA8451_read(MMA8451_ADDR,0x2B);  //read ctrl_reg2
305:
306:   value=(value & 0b11100111) | 0b10010000 ;     //Set high res mode bits, clear te
    est-mode
307:   MMA8451_write(MMA8451_ADDR,0x2b,value);       //write new ctrl_reg2
308:   MMA8451_write(MMA8451_ADDR,0x0e,0x00);        //Range +/- 2g
309:   //MMA8451_write(MMA8451_ADDR,0x31,0x00);        //Cal_Z offset
310:   MMA8451_write(MMA8451_ADDR,0x2a,0b10110101); //6.25hz,low noise, + goto active
     mode
311: }
312:
313: ////////////////////////////////////////////////////////////////////////
314: unsigned char selfTest(){
315:   long int ax,ay,az;
316:   long int x,y,z,xt,yt,zt;
317:   unsigned char i;
318:   ax=ay=az=0L;
319:   //average 10 readings before self test
320:   asm CLRWDT;
321:   for (i=0;i<10;i++) {
322:       read_xyz(0);
323:       ax = ax + (long int) xaccel;
324:       ay = ay + (long int) yaccel;
325:       az = az + (long int) zaccel;
326:       output_xyz();
327:   }
328:   asm CLRWDT;
329:   testMode();
330:   read_xyz(0);  //we'll ignore the first reading
331:   //get the mean of the before self test readings
332:   x = ax;
333:   y = ay;
334:   z = az;
335:   //average 10 self test readings
336:   ax=ay=az=0L;
337:   for (i=0;i<10;i++) {
338:       read_xyz(0);
339:       ax = ax + (long int) xaccel;
340:       ay = ay + (long int) yaccel;
341:       az = az + (long int) zaccel;
342:       uart1_write('T');
343:       output_xyz();
344:   }
345:   asm CLRWDT;
346:   activate();  //Leave self test mode
347:   read_xyz(0);  //we'll ignore the first reading
348:   //Get the mean of the self test readings
349:   xt = ax/10;
350:   yt = ay/10;
351:   zt = az/10;
352:   //average another 10 readings to bracket the self test readings
353:   ax=ay=az=0L;
354:   for (i=0;i<10;i++) {
355:       read_xyz(0);
356:       ax = ax + (long int) xaccel;
357:       ay = ay + (long int) yaccel;
358:       az = az + (long int) zaccel;
359:       output_xyz();
360:   }
361:   asm CLRWDT;
362:   read_xyz(0);
```

```
363:   //get the mean of the before and after self test readings
364:   x = (x + ax)/20;
365:   y = (y + ay)/20;
366:   z = (z + az)/20;
367:   //Now compare
368:   //print the results
369: #define X_SELF_TEST 344
370: #define ST_DX 50
371: #define Y_SELF_TEST 512
372: #define ST_DY 150
373: #define Z_SELF_TEST 3560
374: #define ST_DZ 200
375:   xaccel = (int) x;
376:   yaccel = (int) y;
377:   zaccel = (int) z;
378:   uart1_write_text("N AV: '");
379:   output_xyz();
380:   xaccel = (int) xt;
381:   yaccel = (int) yt;
382:   zaccel = (int) zt;
383:   uart1_write_text("T AV: ");
384:   output_xyz();
385:   xaccel= (int) xt - (int) x;
386:   yaccel= (int) yt - (int) y;
387:   zaccel= (int) zt - (int) z;
388:   uart1_write_text("T-N: ");
389:   output_xyz();
390:   if (( xaccel < X_SELF_TEST - ST_DX) || ( xaccel > X_SELF_TEST + ST_DX) ) return
      (1);
391:   if (( yaccel < Y_SELF_TEST - ST_DY) || ( yaccel > Y_SELF_TEST + ST_DY) ) return
      (1);
392:   if (( zaccel < Z_SELF_TEST - ST_DZ) || ( zaccel > Z_SELF_TEST + ST_DZ) ) return
      (1);
393:   return(0);
394: }
395:
396: ////////////////////////////////////////////////////////////////////////
397: void ljustify(char *txt) {
398:      unsigned short i,j;
399:      i=0; j=0;
400:      while (txt[i]==' ') i++;
401:      while (txt[i]!=0) txt[j++]=txt[i++];
402:      txt[j]=0;
403: }
404: ////////////////////////////////////////////////////////////////////////
405: void output_xyz() {
406:      unsigned char p;
407:      strcpy(&txt[0],"S: ");
408:      p=strlen(txt);
409:      byteToHex( accelbuf[0],&txt[p]);
410:      ljustify(&txt[p]);
411:
412:      strcat(txt,", X: ");
413:      p=strlen(txt);
414:      intToStr( xaccel,&txt[p]);
415:      ljustify(&txt[p]);
416:
417:      strcat(txt,", Y: ");
418:      p=strlen(txt);
419:      intToStr( yaccel,&txt[p]);
420:      ljustify(&txt[p]);
421:
```

```
422:      strcat(txt,", Z: ");
423:      p=strlen(txt);
424:      intToStr( zaccel,&txt[p]);
425:      ljustify(&txt[p]);
426:
427:      uart1_write_text(txt);
428:      uart1_write_text("\r\n");
429:
430: }
431: ////////////////////////////////////////////////////////////////////////////
432: /// Start communicating with MMA8451 and set initial address
433: ////////////////////////////////////////////////////////////////////////////
434: unsigned char MMA8451_start(unsigned short MMA8451_Addr, unsigned char mem_addr)
     {
435:   unsigned short byte;
436:   do {
437:      Soft_I2C_Start();              // issue I2C start signal
438:      byte=Soft_I2C_Write(MMA8451_Addr); // send byte via I2C  (device address + W
      (W=0))
439:      if (byte!=0) {                 //I2C could be busy
440:         Vdelay_ms(1);
441:         MMA8451_error++;;
442:         if (MMA8451_error>10) {Soft_I2C_Stop();return(1);}
443:         }
444:   } while (byte!=0);
445:   if (Soft_I2C_Write(mem_addr)){Soft_I2C_Stop();return(1);}
446:   return(0);
447: }
448:
449: ////////////////////////////////////////////////////////////////////////////
450: ///  Read a single byte from the MMA8451
451: ////////////////////////////////////////////////////////////////////////////
452: char MMA8451_read(unsigned short MMA8451_Addr, unsigned char mem_addr) {
453:   char byte;
454:   MMA8451_error=0;
455:   byte=0;
456:   //Loop till we read without an error
457:   do {
458:     byte=MMA8451_start(MMA8451_addr,mem_addr);
459:     if (byte==0) {
460:        Soft_I2C_Start();     // issue I2C signal repeated start
461:        byte=Soft_I2C_Write(MMA8451_Addr | 0x01);
462:        }
463:   } while ( byte!=0 );           // send byte (device address + R  (R=1))
464:   //Now read the byte
465:   byte = Soft_I2C_Read(0u);       // Read the data (NO acknowledge)
466:   Soft_I2C_Stop();
467:   return (byte);
468: }
469: ////////////////////////////////////////////////////////////////////////////
     ////
470: /// Read bsize bytes from the MMA8451
471: ////////////////////////////////////////////////////////////////////////////
     ////
472: char MMA8451_readblk(unsigned short MMA8451_Addr, unsigned int mem_addr, char *buf
     ffer,unsigned short bsize) {
473:   char byte;
474:   register int ack;
475:   MMA8451_error=0;
476:   byte=0;
477:   //Loop till we setup the read without an error
478:   do {
```

```
479:      byte=MMA8451_start(MMA8451_addr,mem_addr);
480:      Soft_I2C_Start();     // issue I2C signal repeated start
481:      byte=Soft_I2C_Write(MMA8451_Addr | 0x01); // send byte (device address + R  (
     (R=1))
482:      if (byte!=0) {Soft_I2C_Stop();}
483:    } while (byte!=0);
484:
485:    //Now read the data
486:    byte=0;
487:    ack=1;
488:    while (byte<bsize ) {
489:      if (byte+1==bsize) ack=0;         //ack=0 for last read
490:      buffer[byte++] = Soft_I2C_Read(ack);       // Read the data (with acknowledge
     e)
491:    }
492:    Soft_I2C_Stop();
493:    return (0);
494: }
495: ////////////////////////////////////////////////////////////////////////////////
     ///
496: ///  Write a single byte to the eprom
497: ////////////////////////////////////////////////////////////////////////////////
     /
498: char MMA8451_write(unsigned short MMA8451_Addr, unsigned int mem_addr, char edata
     ) {
499:    char byte;
500:    MMA8451_error=0;
501:    byte=0;
502:    if (MMA8451_start(MMA8451_addr,mem_addr)) return(1);
503:    if (Soft_I2C_Write(edata)) {Soft_I2C_Stop();return(1);}             // send data
     a (data to be written)
504:    Soft_I2C_Stop();               // issue I2C stop signal
505:    return(0);
506: }
```