```
rem ************************************************
rem *** at25df641, 64-Mbit SPI Serial Flash Memory ***
rem ************************************************
//
sub at25df641_init
  rem call this to initialize an at25df641 SPI Serial Flash Memory
  rem you must provide a sub at25df641_cs as a callback that will
  rem manipulate the at25df641 cs* pin
  dim cmd as byte, id
  cmd = 0x9f  // Read Manufacturer and Device ID
  gosub at25df641_cs 0
  qspi cmd, id
  gosub at25df641_cs 1
  assert id!=0&&id!=-1
endsub
//
sub at25df641_status status
  rem call this to read the status register of an at25df641 SPI
  rem Serial Flash Memory
  dim cmd as byte, statusb as byte
  cmd = 0x05  // Read Status Register
  gosub at25df641_cs 0
  qspi cmd, statusb
  gosub at25df641_cs 1
  status = statusb
endsub
//
sub at25df641_chip_erase
  rem call this to erase the entire at25df641 SPI Serial Flash Memory
  dim cmd as byte, status
  cmd = 0x06  // Write Enable
  gosub at25df641_cs 0
  qspi cmd
  gosub at25df641_cs 1
  gosub at25df641_status status
  assert (status&3)==2
  cmd = 0x60  // Chip Erase
  gosub at25df641_cs 0
  qspi cmd
  gosub at25df641_cs 1
  do
    gosub at25df641_status status
  until !(status&1)
  assert !status
endsub
//
sub at25df641_4k_erase addr
  rem call this to erase a 4k page of an at25df641 SPI Serial Flash
  rem Memory starting at addr
  dim cmd as byte, a1 as byte, a2 as byte, a3 as byte, status
  cmd = 0x06  // Write Enable
  gosub at25df641_cs 0
  qspi cmd
  gosub at25df641_cs 1
  gosub at25df641_status status
  assert (status&3)==2
  cmd = 0x20  // Block Erase (4-KBytes)
  a1 = addr>>16, a2 = addr>>8, a3 = addr
  gosub at25df641_cs 0
  qspi cmd, a1, a2, a3
  gosub at25df641_cs 1
```

```
  do
    gosub at25df641_status status
  until !(status&1)
  assert !status
endsub
//
sub at25df641_read addr, data
  rem call this to read an arbitrary amount of data from an
  rem at25df641 SPI Serial Flash Memory starting at addr
  dim cmd as byte, a1 as byte, a2 as byte, a3 as byte
  cmd = 0x03  // Read Array
  a1 = addr>>16, a2 = addr>>8, a3 = addr
  gosub at25df641_cs 0
  qspi cmd, a1, a2, a3, data
  gosub at25df641_cs 1
endsub
//
sub at25df641_write addr, data
  rem call this to write an arbitrary amount of data to an
  rem at25df641 SPI Serial Flash Memory starting at addr
  dim cmd as byte, a1 as byte, a2 as byte, a3 as byte, status
  cmd = 0x06  // Write Enable
  gosub at25df641_cs 0
  qspi cmd
  gosub at25df641_cs 1
  gosub at25df641_status status
  assert (status&3)==2
  cmd = 0x02  // Byte/Page Program (1 to 256 Bytes)
  a1 = addr>>16, a2 = addr>>8, a3 = addr
  gosub at25df641_cs 0
  qspi cmd, a1, a2, a3, data
  gosub at25df641_cs 1
  do
    gosub at25df641_status status
  until !(status&1)
  assert !status
endsub
//
// ****************************************
// *** fslbot, Freescale Mechatronics Robot ***
// ****************************************
//
sub fslbot_init
  rem call this to initialize the facial LEDs of the fslbot
  gosub pca9554_init 0x38, 0xfe, 0
endsub
//
sub fslbot_right_step
  rem call this to step the right foot
  lfoot = 0  // relax left foot
  for rfoot = rfoot to 1500+flower step 10  // lean left
    sleep delay ms
  next
  for lfoot = 1500+fstart to 1500+fraise step 10  // stand left
    sleep delay ms
  next
  do   // take right step
    if rfoot>1500+ftip then
      rfoot = rfoot-10
    endif
    if rhip>1500-hshift then
```

```
      rhip = rhip-10
      lhip = rhip
    endif
    sleep delay ms
  until rfoot<=1500+fstart&&rhip<=1500-hshift
  for lfoot = lfoot to 1500 step -10  // fall back right
    sleep delay ms
  next
endsub
//
sub fslbot_left_step
  rem call this to step the left foot
  rfoot = 0  // relax right foot
  for lfoot = lfoot to 1500-flower step -10  // lean right
    sleep delay ms
  next
  for rfoot = 1500-fstart to 1500-fraise step -10  // stand right
    sleep delay ms
  next
  do   // take left step
    if lfoot<1500-ftip then
      lfoot = lfoot+10
    endif
    if lhip<1500+hshift then
      lhip = lhip+10
      rhip = lhip
    endif
    sleep delay ms
  until lfoot>=1500-fstart&&lhip>=1500+hshift
  for rfoot = rfoot to 1500 step 10  // fall back left
    sleep delay ms
  next
endsub
//
sub fslbot_stand_square
  rem call this to stand square
  rfoot = 1500, rhip = 1500, lhip = 1500, lfoot = 1500
endsub
//
sub fslbot_mouth leds
  rem call this to update the facial LEDs of the fslbot
  gosub pca9554_write 0x38, 0xfe, leds
endsub
//
rem ************************************************
rem *** mag3110, Three-Axis, Digital Magnetometer ***
rem ************************************************
//
sub mag3110_init addr
  rem initialize the magnetometer to take 10Hz measurements
  dim reg as byte, data as byte
  // make sure we're talking to the 3110
  reg = 7  // who_am_i
  i2c start addr
  i2c write reg
  i2c read data
  i2c stop
  assert data==0xc4
  // initialize the 3110
  restore mag3110_init_data
  while 1 do
```

```
    read reg, data
    if reg==255 then
      break
    endif
    i2c start addr
    i2c write reg, data
    i2c stop
  endwhile
  label mag3110_init_data
  data 0x11, 0x80  // enable automatic resets
  data 0x10, 0x69  // 10Hz
  data 255, 255
endsub
//
sub mag3110_poll addr, xa, ya, deg
  rem poll the magnetometer x and y values and return the heading
  dim reg as byte, data as byte, x as short, y as short
  reg = 1  // out_x_msb, out_x_lsb, out_y_msb, out_y_lsb
  i2c start addr
  i2c write reg
  i2c read x, y
  i2c stop
  // sign extend the x and y values
  xa = x, ya = y
  if x&0x8000 then
    xa = xa|0xffff0000
  endif
  if y&0x8000 then
    ya = ya|0xffff0000
  endif
  // calibrate the x and y values based on mix/max
  gosub mag3110_cal xa, ya
  // compute and print the heading
  gosub mag3110_heading xa, ya, deg
endsub
//
sub mag3110_heading x, y, deg
  rem compute the heading in degrees from x and y values
  dim q, xq, yq, i, s100, rad100, deg100
  // normalize x and y to the first quadrant
  if x>=0&&y>=0 then
    q = 0, xq = x, yq = y
  elseif x<0&&y<0 then
    q = 2, xq = -x, yq = -y
  elseif x<0 then
    q = 3, xq = y, yq = -x
  else
    assert y<0
    q = 1, xq = -y, yq = x
  endif
  assert xq>=0&&yq>=0
  // N.B. we use 100x scale for integers in this routine
  // our arctangent approximation only works for y<=x
  if yq>xq then
    i = 1, s100 = xq*100/yq
  else
    if xq then
      i = 0, s100 = yq*100/xq
    else
      i = 0, s100 = 0  // we must not be calibrated
    endif
```

```
  endif
  // arctangent(s) = s/(1+0.28*s^2)
  rad100 = s100*100/(100+28*s100*s100/100/100)
  if i then
    // arctangent(s) = pi/2 - arctangent(1/s)
    rad100 = 314/2-rad100
  endif
  // convert back to degrees and denormalize the quadrant
  deg100 = 9000*rad100/(314/2)
  if q==1 then
    deg100 = deg100+27000
  elseif q==2 then
    deg100 = deg100+18000
  elseif q==3 then
    deg100 = deg100+9000
  endif
  // N.B. transform magnetic north for mag orientation
  deg = (deg100/100+90)%360
endsub
//
sub mag3110_cal x, y
  rem calibrate the magnetometer min and max x and y values
  dim xmin, xmax, ymin, ymax
  // get the initial calibration values
  gosub mag3110_getcal xmin, xmax, ymin, ymax
  if x<xmin||!xmin then
    xmin = x
  endif
  if x>xmax||!xmax then
    xmax = x
  endif
  if y<ymin||!ymin then
    ymin = y
  endif
  if y>ymax||!ymax then
    ymax = y
  endif
  // set the updated calibration values
  gosub mag3110_setcal xmin, xmax, ymin, ymax
  // calibrate the caller's values
  x = x-(xmax+xmin)/2, y = y-(ymax+ymin)/2
endsub
//
rem *******************************************************
rem *** mma7455, Three Axis Digital Output Accelerometer ***
rem *******************************************************
//
sub mma7455_init addr
  rem initialize the mma7455 at i2c addr
  dim cmd as byte, data as byte
  cmd = 0x16, data = 0x01  // Mode control
  i2c start addr
  i2c write cmd, data
  i2c stop
endsub
//
sub mma7455_poll addr, x, y, z
  rem poll the mma7455 at i2c addr for the x, y, and z values
  dim cmd as byte, xb as byte, yb as byte, zb as byte
  cmd = 0x6  // 8 bits output value X
  i2c start addr
```

```
  i2c write cmd
  i2c read xb, yb, zb
  i2c stop
  x = xb, y = yb, z = zb
endsub
//
// ************************************************************
// *** mpr121, Proximity Capacitive Touch Sensor Controller ***
// ************************************************************
//
sub mpr121_init addr
  rem just follow AN3944: MPR121 Quick Start Guide
  dim i
  dim r as byte, d as byte
  i2c start addr
  for i = 1 to 0x17 step 2
    r = 0x40+i, d = 0xf
    i2c write r, d
    r = 0x41+i, d = 0xa
    i2c write r, d
  next
  restore mpr121_init_data
  do
    read r, d
    i2c write r, d
  until r==0x5e
  i2c stop
  label mpr121_init_data
  data 0x2b, 0x1, 0x2c, 0x1, 0x2d, 0x0, 0x2e, 0x0
  data 0x2f, 0x1, 0x30, 0x1, 0x31, 0xff, 0x32, 0x2
  data 0x5d, 0x4, 0x7b, 0xb, 0x7d, 0x9c, 0x7e, 0x65
  data 0x7f, 0x8c, 0x5e, 0x8c
endsub
//
sub mpr121_poll addr, bits
  rem read and return both bytes of the touch register
  dim r as byte, r0 as byte, r1 as byte
  i2c start addr
  r = 0
  i2c write r
  i2c read r0, r1
  i2c stop
  bits = r1<<8|r0
endsub
//
// ************************************
// *** PCA9554, 8-bit I2C-bus I/O port ***
// ************************************
//
sub pca9554_init addr, mask, input
  rem configure mask bits of the port for input
  dim cmd as byte, data as byte
  cmd = 3
  i2c start addr
  i2c write cmd
  i2c read data
  i2c stop
  cmd = 3, data = (data&~mask)|(input&mask)
  i2c start addr
  i2c write cmd, data
  i2c stop
```

```
endsub
//
sub pca9554_write addr, mask, value
  rem update mask bits of the output port to value
  dim cmd as byte, data as byte
  cmd = 1
  i2c start addr
  i2c write cmd
  i2c read data
  i2c stop
  cmd = 1, data = (data&~mask)|(value&mask)
  i2c start addr
  i2c write cmd, data
  i2c stop
endsub
//
sub pca9554_read addr, value
  rem read the input port
  dim cmd as byte, data as byte
  cmd = 0
  i2c start addr
  i2c write cmd
  i2c read data
  i2c stop
  value = data
endsub
```