# TWR-MECH Board Support Library for StickOS® BASIC

# Main Program Examples:

## Example 1: at25df641, 64-Mbit SPI Serial Flash Memory & PCA9554, 8-bit I2C-bus I/O port

This example erases a 4k page of an at25df641 64-Mbit SPI Serial Flash Memory whose chip select line is controlled thru an PCA9554 8-bit I2C-bus I/O port at i2c address 0x38.

Note that this example relies on a callback in the main program (called by the library) to set and clear the at25df641 chip select line.

10 dim word

20 rem raise cs\* on pca9554 0x38 bit 0

30 gosub pca9554\_init 0x38, 1, 0

40 gosub pca9554\_write 0x38, 1, 1

50 rem initialize, program, and read the spi flash

60 gosub at25df641\_init

70 gosub at25df641\_4k\_erase 0

80 word = 123

90 gosub at25df641\_write 0, word

100 word = 0

110 gosub at25df641\_read 0, word

120 print word

130 end

140 sub at25df641\_cs cs

150 rem set cs\* as requested by at25df641 subs

160 gosub pca9554\_write 0x38, 1, cs

170 endsub

## Example 2: mma7455, Three Axis Digital Output Accelerometer

This program just displays the accelerometer values from the mma7455 at i2c address 0x1d:

10 dim x, y, z

20 gosub mma7455\_init 0x1d

30 while 1 do

40 gosub mma7455\_poll 0x1d, x, y, z

50 print hex "x = ", x, "y =", y, "z =", z

60 sleep 100 ms

70 endwhile

## Example 3: mag3110, Three-Axis, Digital Magnetometer

This example calibrates the mag3110 magnetometer at i2c address 0xe when the robot is slowly rotated, and then prints the heading in degrees from magnetic north 10 times per second. Note that this program stores the magnetometer calibration values in StickOS flash, but could just as easily store it in the atdf641 using the method above.

Note that this example relies on two callbacks in the main program (called by the library) to set and get the magnetometer calibration (min/max) values.

10 // these are our calibration values stored in flash

20 dim xminf as flash, xmaxf as flash, yminf as flash, ymaxf as flash

30 //

40 // these are our shadow calibration values stored in ram for speed

50 dim xminr, xmaxr, yminr, ymaxr

60 //

70 // this is when we last updated flash

80 dim secondsf

90 secondsf = seconds

100 //

110 // this is our interrupt pin indicating measurement complete

120 dim int as pin an3 for digital input

130 //

140 // this is our magnetometer i2c address

150 dim addr

160 addr = 0xe

170 //

180 // this is the calibrated magnetometer heading

190 dim xa, ya, deg

200 //

210 // initialize the 3110 and start taking interrupts

220 gosub mag3110\_init addr

230 on int do gosub loop

240 halt

250 //

260 // poll the magnetometer x and y values and print the heading

270 sub loop

280 gosub mag3110\_poll addr, xa, ya, deg

290 print "x =", xa, "y =", ya, "deg =", deg

300 endsub

310 //

320 // get the magnetometer calibration values

330 sub mag3110\_getcal xmin, xmax, ymin, ymax

340 // if our shadow values are not set...

350 if xminr==0&&xmaxr==0&&yminr==0&&ymaxr==0 then

360 // read flash

370 // N.B. we could just as easily read eeprom or whatever

380 xmin = xminf, xmax = xmaxf, ymin = yminf, ymax = ymaxf

390 else

400 // read shadow

410 xmin = xminr, xmax = xmaxr, ymin = yminr, ymax = ymaxr

420 endif

430 endsub

440 //

450 // set the magnetometer calibration values

460 sub mag3110\_setcal xmin, xmax, ymin, ymax

470 // update shadow

480 xminr = xmin, xmaxr = xmax, yminr = ymin, ymaxr = ymax

490 // if it has been ten seconds since we last updated flash...

500 if seconds>secondsf+10 then

510 // update flash

520 // N.B. we could just as easily update eeprom or whatever

530 xminf = xmin, xmaxf = xmax, yminf = ymin, ymaxf = ymax

540 secondsf = seconds

550 print "update"

560 endif

570 endsub

## Example 4: fslbot, Freescale Mechatronics Robot & mpr121, Proximity Capacitive Touch Sensor Controller & PCA9554, 8-bit I2C-bus I/O port

This example responds to touch commands on the mouth of the robot to blink the mouth LEDs, and then responds to touch commands on the left and right cheeks of the robot to walk the robot starting with the corresponding foot; a touch on the forehead stops the walk. The mpr121 touch sensor is at i2c address 0x5a.

Note that this example relies on a number of variables (including pin variables for the servos) defined by the main program for use in the walking subs, to specify the walking stride. See the subs in the library below.

10 // \*\*\* fslbot demo \*\*\*

20 // touch mouth to blink leds

30 // adjust potentiometer for walk speed

40 // touch left/right cheek to take 6 steps leading from left/right

50 // touch forehead to stop march

60 //

70 dim blink, lit // state of the mouth LEDs

80 dim right, left // counts of steps to take

90 //

100 // \*\*\* initialize our modules \*\*\*

110 gosub mpr121\_init 0x5a

120 gosub fslbot\_init

130 //

140 // \*\*\* configure the mpr121\_isr \*\*\*

150 dim isr as pin irq7\* for digital input inverted

160 on isr do gosub mpr121\_isr

170 //

180 // \*\*\* configure the mouth\_timer \*\*\*

190 configure timer 1 for 300 ms

200 on timer 1 do gosub mouth\_timer

210 //

220 // \*\*\* configure the servo pins \*\*\*

230 dim rfoot as pin dtin0 for servo output

240 dim rhip as pin dtin1 for servo output

250 dim lhip as pin dtin2 for servo output

260 dim lfoot as pin dtin3 for servo output

270 //

280 // \*\*\* configure the walk speed potentiometer

290 dim pot as pin an6 for analog input

300 //

310 // \*\*\* set the walk stride variables \*\*\*

320 dim delay

330 dim flower, fstart, fraise, ftip, hshift

340 delay = 15

350 flower = 600, fstart = 150, fraise = 230, ftip = 50, hshift = 250

360 //

370 // \*\*\* stand square \*\*\*

380 gosub fslbot\_stand\_square

390 //

400 // \*\*\* main loop -- just walk \*\*\*

410 //

420 while 1 do

430 if left>right&&left>0 then

440 // lead with the left foot

450 gosub fslbot\_left\_step

460 left = left-2

470 elseif right>left&&right>0 then

480 // lead with the right foot

490 gosub fslbot\_right\_step

500 right = right-2

510 else

520 gosub fslbot\_stand\_square

530 endif

540 delay = pot\*15/1650

550 endwhile

560 end

570 //

580 sub mpr121\_isr

590 dim bits

600 // get the touch value and respond appropriately

610 gosub mpr121\_poll 0x5a, bits

620 if bits&128 then

630 print "mouth"

640 blink = !blink

650 endif

660 if bits&64 then

670 print "forehead"

680 left = 0, right = 0

690 endif

700 if bits&32 then

710 print "left cheek"

720 left = 6, right = 5

730 endif

740 if bits&16 then

750 print "right cheek"

760 right = 6, left = 5

770 endif

780 endsub

790 //

800 sub mouth\_timer

810 // blink all bits if we're supposed to

820 if blink then

830 lit = ~lit

840 gosub fslbot\_mouth lit

850 endif

860 endsub

# TWR-MECH Board Support Library:

To load the library, type "new" and then "auto" and then paste the library contents below. Then type "renumber 10000" and "save library" to save the library with unique line numbers. The library subs are then accessible to all main programs above. You can type the "subs" command to quickly list all the subs in the library, and then type the "list *subname*" command to list the BASIC lines of individual subs without having to load them into the current program.

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==0x1f480000

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, 0xc

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