**This code upto page number 11 is dual mcu architecture to test wirepas mesh stack**

**vi test.c**

/\* Wirepas Oy licensed under Apache License, Version 2.0

\*

\* See file LICENSE for full license details.

\*

\*/

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#define LOG\_MODULE\_NAME "Test"

#define MAX\_LOG\_LEVEL INFO\_LOG\_LEVEL

#include "logger.h"

#include "wpc.h"

static int passedTestCount = 0;

static int failedTestCount = 0;

static bool setInitialState(app\_role\_t role,

app\_addr\_t id,

net\_addr\_t network\_add,

net\_channel\_t network\_channel,

bool start)

{

WPC\_stop\_stack();

// Wait for stack to stop

usleep(3 \* 1000 \* 1000);

if (WPC\_set\_node\_address(id) != APP\_RES\_OK)

return false;

if (WPC\_set\_role(role) != APP\_RES\_OK)

return false;

if (WPC\_set\_network\_address(network\_add) != APP\_RES\_OK)

return false;

if (WPC\_set\_network\_channel(network\_channel) != APP\_RES\_OK)

return false;

if (start && (WPC\_start\_stack() == APP\_RES\_OK))

return false;

return true;

}

static bool testFactoryReset()

{

app\_res\_e res = WPC\_stop\_stack();

app\_addr\_t add;

if (res != APP\_RES\_STACK\_ALREADY\_STOPPED && res != APP\_RES\_OK)

{

LOGE("Cannot stop stack\n");

return false;

}

if (WPC\_set\_node\_address(0x123456) != APP\_RES\_OK)

{

LOGE("Cannot set node address\n");

return false;

}

if (WPC\_do\_factory\_reset() != APP\_RES\_OK)

{

LOGE("Cannot do factory reset\n");

return false;

}

// Check node address previously set is correctly removed

if (WPC\_get\_node\_address(&add) != APP\_RES\_ATTRIBUTE\_NOT\_SET)

{

LOGE("Node address is still set\n");

return false;

}

return true;

}

static bool setAppconfig(uint8\_t \* config, uint16\_t interval, uint8\_t size)

{

uint8\_t cur\_seq, new\_seq = 0;

uint16\_t cur\_interval;

uint8\_t cur\_config[128];

app\_res\_e res;

if (size > sizeof(cur\_config))

{

LOGE("Specified size (%d) is too big.\n", size);

return false;

}

res = WPC\_get\_app\_config\_data(&cur\_seq, &cur\_interval, cur\_config, sizeof(cur\_confi

g));

if (res == APP\_RES\_OK)

{

new\_seq = cur\_seq + 1;

}

else

{

LOGE("Cannot get current app config seq. Result = 0x%02x.\n", res);

return false;

}

res = WPC\_set\_app\_config\_data(new\_seq, interval, config, size);

if (res != APP\_RES\_OK)

{

LOGE("Cannot set new app config with seq = %d. Result = 0x%02x.\n", new\_seq, re

s);

return false;

}

// Read back App config

res = WPC\_get\_app\_config\_data(&cur\_seq, &cur\_interval, cur\_config, sizeof(cur\_confi

g));

if (res != APP\_RES\_OK)

{

LOGE("Cannot read back new app config. Result = 0x%02x.\n", res);

return false;

}

if (memcmp(config, cur\_config, size) != 0)

{

LOGE("App config set differs from app config read.\n");

return false;

}

return true;

}

static bool dumpCSAPAttributes()

{

uint8\_t role;

net\_channel\_t channel;

app\_addr\_t node\_address;

net\_addr\_t network;

uint8\_t mtu, pdus, scratch\_seq, first\_channel, last\_channel, data\_size;

uint16\_t firmware[4], api\_v, hw\_magic, stack\_profile;

if (WPC\_get\_role(&role) == APP\_RES\_ATTRIBUTE\_NOT\_SET)

{

LOGW("Role not set\n");

role = 0;

}

if (WPC\_get\_node\_address(&node\_address) == APP\_RES\_ATTRIBUTE\_NOT\_SET)

{

LOGW("Node address not set\n");

node\_address = 0;

}

if (WPC\_get\_network\_address(&network) == APP\_RES\_ATTRIBUTE\_NOT\_SET)

{

LOGW("Network address not set\n");

network = 0;

}

if (WPC\_get\_network\_channel(&channel) == APP\_RES\_ATTRIBUTE\_NOT\_SET)

{

LOGW("Network channel not set\n");

channel = 0;

}

LOGI("Role is 0x%02x / Add is %d / Network is 0x%06x (channel = %d)\n", role, node\_

address, network, channel);

WPC\_get\_mtu(&mtu);

LOGI("MTU size is %d\n", mtu);

WPC\_get\_pdu\_buffer\_size(&pdus);

LOGI("PDU size is %d\n", pdus);

WPC\_get\_scratchpad\_sequence(&scratch\_seq);

LOGI("Scratchpad seq is %d\n", scratch\_seq);

WPC\_get\_mesh\_API\_version(&api\_v);

LOGI("Mesh API version is %d\n", api\_v);

WPC\_get\_firmware\_version(firmware);

LOGI("Firmware version is %d:%d:%d:%d\n", firmware[0], firmware[1], firmware[2], fi

rmware[3]);

WPC\_get\_channel\_limits(&first\_channel, &last\_channel);

LOGI("Channel Limits are: [%d - %d]\n", first\_channel, last\_channel);

WPC\_get\_app\_config\_data\_size(&data\_size);

LOGI("Max config data size is %d\n", data\_size);

WPC\_get\_hw\_magic(&hw\_magic);

LOGI("Radio hardware is %d\n", hw\_magic);

WPC\_get\_stack\_profile(&stack\_profile);

LOGI("Stack profile is %d\n", stack\_profile);

return true;

}

static bool testCipherKey()

{

bool key\_set;

if (WPC\_is\_cipher\_key\_set(&key\_set) != APP\_RES\_OK)

{

return false;

}

if (key\_set)

{

LOGW("Key already set\n");

// return false

}

LOGI("No cipher key\n");

uint8\_t key[16] = {

0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0

x0d, 0x0e, 0x0f};

LOGI("Set cipher key\n");

if (WPC\_set\_cipher\_key(key) != APP\_RES\_OK)

{

LOGE("Cannot set key\n");

return false;

}

LOGI("Check cipher key set\n");

if (WPC\_is\_cipher\_key\_set(&key\_set) != APP\_RES\_OK)

{

return false;

}

if (!key\_set)

{

LOGE("Key is not set\n");

return false;

}

LOGI("Remove cipher key\n");

if (WPC\_remove\_cipher\_key() != APP\_RES\_OK)

{

LOGE("Cannot remove key\n");

return false;

}

LOGI("Check cipher key not set\n");

if (WPC\_is\_cipher\_key\_set(&key\_set) != APP\_RES\_OK)

{

return false;

}

if (key\_set)

{

LOGE("Key is still set\n");

return false;

}

return true;

}

static bool testAuthenticationKey()

{

bool key\_set;

if (WPC\_is\_authentication\_key\_set(&key\_set) != APP\_RES\_OK)

{

return false;

}

if (key\_set)

{

LOGW("Key already set\n");

// return false

}

LOGI("No authentication key\n");

uint8\_t key[16] = {

0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0

x0d, 0x0e, 0x0f};

LOGI("Set authentication key\n");

if (WPC\_set\_authentication\_key(key) != APP\_RES\_OK)

{

LOGE("Cannot set key\n");

return false;

}

LOGI("Check authentication key set\n");

if (WPC\_is\_authentication\_key\_set(&key\_set) != APP\_RES\_OK)

{

return false;

}

if (!key\_set)

{

LOGE("Key is not set\n");

return false;

}

LOGI("Remove authentication key\n");

if (WPC\_remove\_authentication\_key() != APP\_RES\_OK)

{

LOGE("Cannot remove key\n");

return false;

}

LOGI("Check authentication key not set\n");

if (WPC\_is\_authentication\_key\_set(&key\_set) != APP\_RES\_OK)

{

return false;

}

if (key\_set)

{

LOGE("Key is still set\n");

return false;

}

return true;

}

static bool testScratchpadTarget()

{

uint8\_t target\_seq;

uint16\_t target\_crc;

uint8\_t action;

uint8\_t param;

// Read it a first time

if (WPC\_read\_target\_scratchpad(&target\_seq, &target\_crc, &action, &param) != APP\_RE

S\_OK)

{

LOGE("Cannot read target scratchpad\n");

return false;

}

LOGI("Target is: %d, 0x%x, %d, %d\n", target\_seq, target\_crc, action, param);

app\_res\_e res = WPC\_write\_target\_scratchpad(12, 0x1234, 2, 13);

if (res != APP\_RES\_OK)

{

LOGE("Cannot write target scratchpad %d\n", res);

return false;

}

LOGI("Write new target\n");

if (WPC\_read\_target\_scratchpad(&target\_seq, &target\_crc, &action, &param))

{

LOGE("Cannot read target back scratchpad\n");

return false;

}

LOGI("Target read back is: %d, 0x%x, %d, %d\n", target\_seq, target\_crc, action, par

am);

if (target\_seq != 12 || target\_crc != 0x1234 || action != 2 || param != 13)

{

LOGE("Wrong read-back value\n");

return false;

}

return true;

}

void onDataSent(uint16\_t pduid, uint32\_t buffering\_delay, uint8\_t result)

{

LOGI("Indication received for %d, delay=%d, result=%d\n", pduid, buffering\_delay, r

esult);

}

static bool testSendWithCallbacks()

{

// Send 5 message to ourself

uint8\_t data[] = "This is a test message #00 with ind\0";

for (int i = 0; i < 2; i++)

{

data[24] = i / 10 + 0x30;

data[25] = i % 10 + 0x30;

if (WPC\_send\_data(data, sizeof(data), i, APP\_ADDR\_ANYSINK, APP\_QOS\_HIGH, 50, 50

, onDataSent, 0) !=

APP\_RES\_OK)

{

return false;

}

}

return true;

}

static bool testSendWithoutCallbacks()

{

// Send 5 message to ourself

uint8\_t data[] = "This is a test message #00\0";

for (int i = 2; i < 4; i++)

{

data[24] = i / 10 + 0x30;

data[25] = i % 10 + 0x30;

if (WPC\_send\_data(data, sizeof(data), i, APP\_ADDR\_ANYSINK, APP\_QOS\_HIGH, 50, 50

, NULL, 0) != APP\_RES\_OK)

{

return false;

}

}

return true;

}

static bool testSendWithInitialTime()

{

// Send 5 message to ourself

uint8\_t data[] = "This is a test message #00\0";

for (int i = 4; i < 6; i++)

{

data[24] = i / 10 + 0x30;

data[25] = i % 10 + 0x30;

if (WPC\_send\_data(data, sizeof(data), i, APP\_ADDR\_ANYSINK, APP\_QOS\_HIGH, 50, 50

, NULL, 300) !=

APP\_RES\_OK)

{

return false;

}

}

return true;

}

static void onAppConfigDataReceived(uint8\_t seq, uint16\_t interval, uint8\_t \* config\_p)

{

LOGI("AppConfig received %d, %d, %s\n", seq, interval, config\_p);

}

static bool testAppConfigData()

{

uint8\_t new\_config[128];

uint8\_t max\_size;

if (WPC\_get\_app\_config\_data\_size(&max\_size) != APP\_RES\_OK || (max\_size > 128))

{

LOGE("Cannot get max app config size of bigger than 128 bytes\n");

return false;

}

for (uint8\_t i = 0; i < max\_size; i++)

{

new\_config[i] = i;

}

if (!setAppconfig(new\_config, 30, max\_size))

{

LOGE("Cannot setAppConfig\n");

return false;

}

if (WPC\_register\_for\_app\_config\_data(onAppConfigDataReceived) != APP\_RES\_OK)

{

LOGE("Cannot register for app\_config\n");

return false;

}

return true;

}

static bool testMSAPAttributesStackOff()

{

uint16\_t min, max;

// Access cycle

int res = WPC\_get\_access\_cycle\_range(&min, &max);

if (res != APP\_RES\_OK && res != APP\_RES\_ATTRIBUTE\_NOT\_SET)

{

LOGE("Cannot get access cycle range\n");

return false;

}

if (res != APP\_RES\_ATTRIBUTE\_NOT\_SET)

{

LOGI("Access cycle range is %d - %d\n", min, max);

}

if (WPC\_set\_access\_cycle\_range(4000, 4000) != APP\_RES\_OK)

{

LOGE("Cannot set cycle range\n");

return false;

}

else

{

if (WPC\_get\_access\_cycle\_range(&min, &max) != APP\_RES\_OK)

{

LOGE("Cannot get cycle range\n");

return false;

}

if (min != 4000 || max != 4000)

{

LOGE("Cannot read back access cycle range set\n");

return false;

}

}

LOGI("Cycle range is now %d - %d\n", min, max);

if (WPC\_get\_access\_cycle\_limits(&min, &max) != APP\_RES\_OK)

{

LOGE("Cannot get cycle limits\n");

return false;

}

LOGI("Cycle range limits is %d - %d\n", min, max);

return true;

}

static bool testMSAPAttributesStackOn()

{

app\_res\_e res;

uint8\_t res8;

uint16\_t res16;

uint32\_t res32;

if (WPC\_get\_stack\_status(&res8) != APP\_RES\_OK)

{

LOGE("Cannot get stack status\n");

return false;

}

LOGI("Stack status is %d\n", res8);

if (WPC\_get\_PDU\_buffer\_usage(&res8) != APP\_RES\_OK)

{

LOGE("Cannot get PDU buffer usage\n");

return false;

}

LOGI("PDU buffer usage is %d\n", res8);

if (WPC\_get\_PDU\_buffer\_capacity(&res8) != APP\_RES\_OK)

{

LOGE("Cannot get PDU buffer capacity\n");

return false;

}

LOGI("PDU buffer capacity is %d\n", res8);

// Battery remaining setting

res = WPC\_get\_remaining\_energy(&res8);

if (res == APP\_RES\_INTERNAL\_ERROR)

{

// Dualmcu API error code 1 while reading attributes

// means "Unsupported attribute id". That is mapped to

// APP\_RES\_INTERNAL\_ERROR. Not ideal error code to

// indicate that the feature has been depricated or

// not implemented in this particular dualmcu implementation.

LOGI("Energy interface not implemented.\n");

}

else if (res != APP\_RES\_OK)

{

LOGE("Cannot get remaining battery usage. Result = 0x%02x.\n", res);

return false;

}

else

{

LOGI("Remaining battery is %d\n", res8);

if (WPC\_set\_remaining\_energy(128) != APP\_RES\_OK)

{

LOGE("Cannot set remaining battery usage\n");

return false;

}

else

{

if (WPC\_get\_remaining\_energy(&res8) != APP\_RES\_OK)

{

LOGE("Cannot get just set remaining battery usage\n");

return false;

}

if (res8 != 128)

{

LOGE("Expecting 128 but receive %d\n", res8);

return false;

}

}

LOGI("Able to set remaining battery to %d\n", res8);

}

if (WPC\_get\_autostart(&res8) != APP\_RES\_OK)

{

LOGE("Cannot get autostart setting\n");

return false;

}

LOGI("Autostart is %d\n", res8);

if (WPC\_get\_route\_count(&res8) != APP\_RES\_OK)

{

LOGE("Cannot get route count\n");

return false;

}

LOGI("Route count is %d\n", res8);

if (WPC\_get\_system\_time(&res32) != APP\_RES\_OK)

{

LOGE("Cannot get system time\n");

return false;

}

LOGI("System time is %d\n", res32);

if (WPC\_get\_current\_access\_cycle(&res16) != APP\_RES\_OK)

{

LOGE("Cannot get current access cycle\n");

return false;

}

LOGI("Current Access Cycle is %d\n", res16);

return true;

}

bool testClearScratchpad()

{

app\_scratchpad\_status\_t status;

if (WPC\_clear\_local\_scratchpad() != APP\_RES\_OK)

{

LOGE("Cannot clear local scratchpad\n");

return false;

}

if (WPC\_get\_local\_scratchpad\_status(&status) != APP\_RES\_OK)

{

LOGE("Cannot get scratchpad status\n");

return false;

}

if (status.scrat\_len != 0)

{

LOGE("Scratchpad is not cleared\n");

return false;

}

return true;

}

#define BLOCK\_SIZE 128

#define SEQ\_NUMBER 50

#define OTAP\_UPLOAD\_FILE\_PATH "otap\_files/dummy.otap"

#define OTAP\_DOWNLOAD\_FILE\_PATH "otap\_files/downloaded.otap"

bool testUploadScratchpad()

{

FILE \* fp;

app\_scratchpad\_status\_t status;

long file\_size = 0;

long written = 0;

const char \* filename = OTAP\_UPLOAD\_FILE\_PATH;

uint8\_t block[BLOCK\_SIZE];

fp = fopen(filename, "rb");

if (fp == NULL)

{

LOGE("Cannot open file %s. Please update OTAP\_UPLOAD\_FILE\_PATH to a "

"valid "

"otap image\n",

filename);

return false;

}

/\* Get size of binary \*/

fseek(fp, 0L, SEEK\_END);

file\_size = ftell(fp);

if (file\_size <= 0)

{

LOGE("Cannot determine file size\n");

return false;

}

LOGI("Uploading otap file of %d bytes\n", file\_size);

/\* Set cursor to beginning of file \*/

fseek(fp, 0L, SEEK\_SET);

/\* Start scratchpad update\*/

if (WPC\_start\_local\_scratchpad\_update(file\_size, SEQ\_NUMBER) != APP\_RES\_OK)

{

LOGE("Cannot start scratchpad update\n");

return false;

}

/\* Send scratchpad image block by block \*/

LOGI("Start sending otap (it may take up to 1 minute)\n");

long remaining = file\_size;

while (remaining > 0)

{

uint8\_t block\_size = (remaining > BLOCK\_SIZE) ? BLOCK\_SIZE : remaining;

size\_t read;

read = fread(block, 1, block\_size, fp);

if (read != block\_size)

{

LOGE("Error while reading file asked = %d read = %d\n", block\_size, read);

break;

}

/\* Send the block \*/

if (WPC\_upload\_local\_block\_scratchpad(block\_size, block, written) != APP\_RES\_OK

)

{

LOGE("Cannot upload scratchpad block\n");

break;

}

written += block\_size;

remaining -= block\_size;

}

if (remaining > 0)

{

return false;

}

if (WPC\_get\_local\_scratchpad\_status(&status) != APP\_RES\_OK)

{

LOGE("Cannot get scratchpad status\n");

return false;

}

if (status.scrat\_len != file\_size)

{

LOGE("Scratchpad is not uploaded correctly (wrong size) %d vs %d\n",

status.scrat\_len,

file\_size);

return false;

}

if (status.scrat\_seq\_number != SEQ\_NUMBER)

{

LOGE("Wrong seq number after uploading a scratchpad image \n");

return false;

}

return true;

}

bool testDownloadScratchpad()

{

FILE \* fp;

uint32\_t scratchpad\_size;

long read = 0;

const char \* filename = OTAP\_DOWNLOAD\_FILE\_PATH;

uint8\_t block[BLOCK\_SIZE];

fp = fopen(filename, "wb");

if (fp == NULL)

{

LOGE("Cannot open file %s. Please update OTAP\_DOWNLOAD\_FILE\_PATH to a "

"valid path\n",

filename);

return false;

}

/\* Get size of scratchpad \*/

if (WPC\_get\_scratchpad\_size(&scratchpad\_size) != APP\_RES\_OK)

{

LOGE("Cannot read scratchpad size\n");

return false;

}

LOGI("Downloading otap file of %d bytes\n", scratchpad\_size);

/\* Receive scratchpad image block by block \*/

LOGI("Start receiving otap (it may take up to 1 minute)\n");

uint32\_t remaining = scratchpad\_size;

while (remaining > 0)

{

uint8\_t block\_size = (remaining > BLOCK\_SIZE) ? BLOCK\_SIZE : remaining;

/\* Receive the block \*/

const app\_res\_e download\_res = WPC\_download\_local\_scratchpad(block\_size, block,

read);

if (download\_res != APP\_RES\_OK)

{

LOGE("Cannot download scratchpad block\n");

if (download\_res == APP\_RES\_ACCESS\_DENIED)

{

LOGE("Access denied when downloading scratchpad block. "

"Make sure the dual MCU application was built with scratchpad read

ing enabled.\n");

}

break;

}

size\_t written = fwrite(block, 1, block\_size, fp);

if (written != block\_size)

{

LOGE("Error while writing file asked = %d read = %d\n", block\_size, written

);

break;

}

read += block\_size;

remaining -= block\_size;

}

fclose(fp);

if (remaining > 0)

{

return false;

}

LOGI("Please compare files %s and %s\n", OTAP\_UPLOAD\_FILE\_PATH, OTAP\_DOWNLOAD\_FILE\_

PATH);

return true;

}

static bool scan\_done = false;

static void onScanNeighborsDone(uint8\_t status)

{

LOGI("Scan neighbors is done res=%d\n", status);

scan\_done = true;

}

static bool testScanNeighbors()

{

app\_nbors\_t neighbors\_list;

// Register for end of scan Neighbors

if (WPC\_register\_for\_scan\_neighbors\_done(onScanNeighborsDone) != APP\_RES\_OK)

{

LOGE("Cannot register for remote status \n");

return false;

}

// Ask for a scan

if (WPC\_start\_scan\_neighbors() != APP\_RES\_OK)

{

LOGE("Cannot start scan\n");

return false;

}

LOGI("Wait 5 seconds for scan result\n");

usleep(5 \* 1000 \* 1000);

// scan\_done should be protected but we can assume

// that 5 sec is enough and no race can occur

if (!scan\_done)

{

LOGE("Scan is not done\n");

return false;

}

if (WPC\_get\_neighbors(&neighbors\_list) != APP\_RES\_OK)

{

LOGE("Cannot get neighbors list\n");

return false;

}

LOGI("Get neighbors done and node has %d neighbors\n", neighbors\_list.number\_of\_nei

ghbors);

if (neighbors\_list.number\_of\_neighbors > 0)

{

LOGI("First node: %d, ch=%d, cost=%d, link=%d, type=%d, rssi=%d, "

"tx\_power=%d, rx\_power=%d, last\_update=%d \n",

neighbors\_list.nbors[0].add,

neighbors\_list.nbors[0].channel,

neighbors\_list.nbors[0].cost,

neighbors\_list.nbors[0].link\_rel,

neighbors\_list.nbors[0].nbor\_type,

neighbors\_list.nbors[0].norm\_rssi,

neighbors\_list.nbors[0].tx\_power,

neighbors\_list.nbors[0].rx\_power,

neighbors\_list.nbors[0].last\_update);

}

if (WPC\_unregister\_from\_scan\_neighbors\_done() != APP\_RES\_OK)

{

LOGE("Cannot unregister from scan neighbors \n");

return false;

}

return true;

}

static void onStackStatusReceived(uint8\_t status)

{

LOGI("Stack status received %d\n", status);

}

static bool testStackStatus()

{

if (WPC\_register\_for\_stack\_status(onStackStatusReceived) != APP\_RES\_OK)

{

return false;

}

return true;

}

// Macro to launch a test and check result

#define RUN\_TEST(\_test\_func\_, \_expected\_result\_) \

do \

{ \

LOGI("### Starting test %s\n", #\_test\_func\_); \

if (\_test\_func\_() != \_expected\_result\_) \

{ \

LOGE("### Test is FAIL\n\n"); \

failedTestCount++; \

} \

else \

{ \

LOGI("### Test is PASS\n\n"); \

passedTestCount++; \

} \

} while (0)

int GetPassedTestCount()

{

return passedTestCount;

}

int GetFailedTestCount()

{

return failedTestCount;

}

int Test\_runAll()

{

uint8\_t status;

RUN\_TEST(testFactoryReset, true);

setInitialState(APP\_ROLE\_SINK, 1234, 0x654321, 5, false);

RUN\_TEST(testScratchpadTarget, true);

RUN\_TEST(dumpCSAPAttributes, true);

RUN\_TEST(testAuthenticationKey, true);

RUN\_TEST(testCipherKey, true);

RUN\_TEST(testMSAPAttributesStackOff, true);

RUN\_TEST(testStackStatus, true);

// Start the stack for following tests

WPC\_start\_stack();

// Ensure stack is started to avoid wrong test result

while (true)

{

if (WPC\_get\_stack\_status(&status) != APP\_RES\_OK)

{

LOGI("Cannot read stack state\n");

continue;

}

if (status == 0)

{

break;

}

LOGI("Waiting for stack start\n");

}

RUN\_TEST(testSendWithCallbacks, true);

RUN\_TEST(testSendWithoutCallbacks, true);

RUN\_TEST(testSendWithInitialTime, true);

RUN\_TEST(testAppConfigData, true);

RUN\_TEST(testMSAPAttributesStackOn, true);

RUN\_TEST(testScanNeighbors, true);

return 0;

}

int Test\_scratchpad()

{

// Configure node as a sink

setInitialState(APP\_ROLE\_SINK, 1234, 0x654321, 5, false);

// Set app config

setAppconfig((uint8\_t \*) "Test scratchpad", 1800, 14);

RUN\_TEST(testClearScratchpad, true);

RUN\_TEST(testUploadScratchpad, true);

RUN\_TEST(testDownloadScratchpad, true);

// Start the stack for following tests

WPC\_start\_stack();

return 0;

}

**vi test.h**

/\* Wirepas Oy licensed under Apache License, Version 2.0

\*

\* See file LICENSE for full license details.

\*

\*/

/\*\*

\* \brief Run all the tests

\*/

int Test\_runAll();

/\*\*

\* \brief Test scratchpad functionality

\*/

int Test\_scratchpad();

/\*\*

\* \brief Returns number of passed test cases

\*/

int GetPassedTestCount();

/\*\*

\* \brief Returns number of failed test cases

\*/

int GetFailedTestCount();

**vi app.c**

**/**\* Wirepas Oy licensed under Apache License, Version 2.0

\*

\* See file LICENSE for full license details.

\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include "test.h"

#include "wpc.h"

#define LOG\_MODULE\_NAME "TestApp"

#define MAX\_LOG\_LEVEL INFO\_LOG\_LEVEL

#include "logger.h"

// Default serial port

char \* port\_name = "/dev/ttyACM0";

int main(int argc, char \* argv[])

{

unsigned long bitrate = DEFAULT\_BITRATE;

if (argc > 1)

{

port\_name = argv[1];

}

if (argc > 2)

{

bitrate = strtoul(argv[2], NULL, 0);

}

if (WPC\_initialize(port\_name, bitrate) != APP\_RES\_OK)

return -1;

Test\_runAll();

Test\_scratchpad();

WPC\_close();

LOGI("Test summary: Passed: %d, Failed: %d\n", GetPassedTestCount(), GetFailedTestC

ount());

return GetFailedTestCount();

}

//from here i have changed to single mcu

**test1.c**

**/\* Wirepas Oy licensed under Apache License, Version 2.0**

**\***

**\* See file LICENSE for full license details.**

**\***

**\*/**

**#include <stdio.h>**

**#include <string.h>**

**#include <unistd.h>**

**#define LOG\_MODULE\_NAME "Test"**

**#define MAX\_LOG\_LEVEL INFO\_LOG\_LEVEL**

**#include "logger.h"**

**#include "api.h" // Include the Single-MCU API header**

**static int passedTestCount = 0;**

**static int failedTestCount = 0;**

**static bool setInitialState(app\_role\_t role,**

**app\_addr\_t id,**

**net\_addr\_t network\_add,**

**net\_channel\_t network\_channel,**

**bool start)**

**{**

**lib\_settings->stopStack();**

**// Wait for stack to stop**

**usleep(3 \* 1000 \* 1000);**

**if (lib\_settings->setNodeAddress(id) != APP\_RES\_OK)**

**return false;**

**if (lib\_settings->setNodeRole(role) != APP\_RES\_OK)**

**return false;**

**if (lib\_settings->setNetworkAddress(network\_add) != APP\_RES\_OK)**

**return false;**

**if (lib\_settings->setNetworkChannel(network\_channel) != APP\_RES\_OK)**

**return false;**

**if (start && (lib\_settings->startStack() != APP\_RES\_OK))**

**return false;**

**return true;**

**}**

**static bool testFactoryReset()**

**{**

**app\_res\_e res = lib\_settings->stopStack();**

**app\_addr\_t add;**

**if (res != APP\_RES\_STACK\_ALREADY\_STOPPED && res != APP\_RES\_OK)**

**{**

**LOGE("Cannot stop stack\n");**

**return false;**

**}**

**if (lib\_settings->setNodeAddress(0x123456) != APP\_RES\_OK)**

**{**

**LOGE("Cannot set node address\n");**

**return false;**

**}**

**if (lib\_settings->factoryReset() != APP\_RES\_OK)**

**{**

**LOGE("Cannot do factory reset\n");**

**return false;**

**}**

**// Check node address previously set is correctly removed**

**if (lib\_settings->getNodeAddress(&add) != APP\_RES\_ATTRIBUTE\_NOT\_SET)**

**{**

**LOGE("Node address is still set\n");**

**return false;**

**}**

**return true;**

**}**

**// Other test functions remain similar but replace `WPC\_\*` calls with `lib\_\*` API calls**

**// For example, replace `WPC\_send\_data` with `lib\_data->sendData`**

**// Replace `WPC\_get\_neighbors` with `lib\_settings->getNeighbors`**

**// Macro to launch a test and check result**

**#define RUN\_TEST(\_test\_func\_, \_expected\_result\_) \**

**do \**

**{ \**

**LOGI("### Starting test %s\n", #\_test\_func\_); \**

**if (\_test\_func\_() != \_expected\_result\_) \**

**{ \**

**LOGE("### Test is FAIL\n\n"); \**

**failedTestCount++; \**

**} \**

**else \**

**{ \**

**LOGI("### Test is PASS\n\n"); \**

**passedTestCount++; \**

**} \**

**} while (0)**

**int GetPassedTestCount()**

**{**

**return passedTestCount;**

**}**

**int GetFailedTestCount()**

**{**

**return failedTestCount;**

**}**

**int Test\_runAll()**

**{**

**RUN\_TEST(testFactoryReset, true);**

**setInitialState(APP\_ROLE\_SINK, 1234, 0x654321, 5, false);**

**// Add other tests here, replacing WPC\_\* calls with lib\_\* API calls**

**return 0;**

**}**

**int Test\_scratchpad()**

**{**

**// Configure node as a sink**

**setInitialState(APP\_ROLE\_SINK, 1234, 0x654321, 5, false);**

**// Add scratchpad-related tests here**

**return 0;**

**}**

**test1.h**

**as same as already exists**

**app1.c**

/\* Wirepas Oy licensed under Apache License, Version 2.0

\*

\* See file LICENSE for full license details.

\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include "test.h"

#include "api.h" // Include the Single-MCU API header

#define LOG\_MODULE\_NAME "TestApp"

#define MAX\_LOG\_LEVEL INFO\_LOG\_LEVEL

#include "logger.h"

int main(int argc, char \* argv[])

{

// Initialize the Wirepas Mesh stack

if (lib\_settings->init() != APP\_RES\_OK)

{

LOGE("Failed to initialize the stack\n");

return -1;

}

Test\_runAll();

Test\_scratchpad();

lib\_settings->stopStack();

LOGI("Test summary: Passed: %d, Failed: %d\n", GetPassedTestCount(), GetFailedTestCount());

return GetFailedTestCount();

}