***Appendix***

**A.** Data Sheets

1. PIC 32MX

2. Flash Memory

3. Power Saving Modes

4. Watchdog Timer and Power-up Timer

5. NRF80001

6. UV Light Sensor

7. Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers

8. Low Noise, Positive-Regulated Charge Pump

9. UV-B Sensor

10. LI-POLYMER BATTERY

 **B.** Software listing:

**i.** Source Files:

1. aci\_queue.c

#include "hal\_aci\_tl.h"

#include "aci\_queue.h"

#include "ble\_assert.h"

#include <stdlib.h>

#include <stdbool.h>

#include <string.h>

#include <stdbool.h>

//#define true 1

//#define false 0

//#define bool \_Bool

void aci\_queue\_init(aci\_queue\_t \*aci\_q)

{

 uint8\_t loop;

 ble\_assert(NULL != aci\_q);

 aci\_q->head = 0;

 aci\_q->tail = 0;

 for(loop=0; loop<ACI\_QUEUE\_SIZE; loop++)

 {

 aci\_q->aci\_data[loop].buffer[0] = 0x00;

 aci\_q->aci\_data[loop].buffer[1] = 0x00;

 }

}

bool aci\_queue\_dequeue(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data)

{

 ble\_assert(NULL != aci\_q);

 ble\_assert(NULL != p\_data);

 if (aci\_queue\_is\_empty(aci\_q))

 {

 return false;

 }

 memcpy((uint8\_t \*)p\_data, (uint8\_t \*)&(aci\_q->aci\_data[aci\_q->head % ACI\_QUEUE\_SIZE]), sizeof(hal\_aci\_data\_t));

 ++aci\_q->head;

 return true;

}

//bool aci\_queue\_dequeue\_from\_isr(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data)

//{

 //ble\_assert(NULL != aci\_q);

 //ble\_assert(NULL != p\_data);

 //if (aci\_queue\_is\_empty\_from\_isr(aci\_q))

 //{

 // return false;

 //}

 //memcpy((uint8\_t \*)p\_data, (uint8\_t \*)&(aci\_q->aci\_data[aci\_q->head % ACI\_QUEUE\_SIZE]), sizeof(hal\_aci\_data\_t));

 //++aci\_q->head;

 //return true;

//}

bool aci\_queue\_enqueue(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data)

{

 const uint8\_t length = p\_data->buffer[0];

 ble\_assert(NULL != aci\_q);

 ble\_assert(NULL != p\_data);

 if (aci\_queue\_is\_full(aci\_q))

 {

 return false;

 }

 aci\_q->aci\_data[aci\_q->tail % ACI\_QUEUE\_SIZE].status\_byte = 0;

 memcpy((uint8\_t \*)&(aci\_q->aci\_data[aci\_q->tail % ACI\_QUEUE\_SIZE].buffer[0]), (uint8\_t \*)&p\_data->buffer[0], length + 1);

 ++aci\_q->tail;

 return true;

}

//bool aci\_queue\_enqueue\_from\_isr(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data)

//{

 //const uint8\_t length = p\_data->buffer[0];

 //ble\_assert(NULL != aci\_q);

 //ble\_assert(NULL != p\_data);

 //if (aci\_queue\_is\_full\_from\_isr(aci\_q))

 //{

 // return false;

 //}

 //aci\_q->aci\_data[aci\_q->tail % ACI\_QUEUE\_SIZE].status\_byte = 0;

 //memcpy((uint8\_t \*)&(aci\_q->aci\_data[aci\_q->tail % ACI\_QUEUE\_SIZE].buffer[0]), (uint8\_t \*)&p\_data->buffer[0], length + 1);

 // ++aci\_q->tail;

 //return true;

//}

bool aci\_queue\_is\_empty(aci\_queue\_t \*aci\_q)

{

 bool state = false;

 ble\_assert(NULL != aci\_q);

 //Critical section

 if (aci\_q->head == aci\_q->tail)

 {

 state = true;

 }

 return state;

}

//bool aci\_queue\_is\_empty\_from\_isr(aci\_queue\_t \*aci\_q)

//{

 //ble\_assert(NULL != aci\_q);

 //return aci\_q->head == aci\_q->tail;

//}

bool aci\_queue\_is\_full(aci\_queue\_t \*aci\_q)

{

 bool state;

 ble\_assert(NULL != aci\_q);

 //This should be done in a critical section

 state = (aci\_q->tail == aci\_q->head + ACI\_QUEUE\_SIZE);

 //end

 return state;

}

//bool aci\_queue\_is\_full\_from\_isr(aci\_queue\_t \*aci\_q)

//{

 //ble\_assert(NULL != aci\_q);

 //return (aci\_q->tail == aci\_q->head + ACI\_QUEUE\_SIZE);

//}

bool aci\_queue\_peek(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data)

{

 ble\_assert(NULL != aci\_q);

 ble\_assert(NULL != p\_data);

 if (aci\_queue\_is\_empty(aci\_q))

 {

 return false;

 }

 memcpy((uint8\_t \*)p\_data, (uint8\_t \*)&(aci\_q->aci\_data[aci\_q->head % ACI\_QUEUE\_SIZE]), sizeof(hal\_aci\_data\_t));

 return true;

}

//bool aci\_queue\_peek\_from\_isr(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data)

//{

 //ble\_assert(NULL != aci\_q);

 //ble\_assert(NULL != p\_data);

 //if (aci\_queue\_is\_empty\_from\_isr(aci\_q))

 //{

 //return false;

 //}

 //memcpy((uint8\_t \*)p\_data, (uint8\_t \*)&(aci\_q->aci\_data[aci\_q->head % ACI\_QUEUE\_SIZE]), sizeof(hal\_aci\_data\_t));

 //return true;

//}

2. aci\_setup.c

#include "lib\_aci.h"

#include "aci\_setup.h"

#include <stdbool.h>

#include <stdio.h>

//#define true 1

//#define false 0

//#define bool \_Bool // need this since converting from C++

// aci\_struct that will contain

// total initial credits

// current credit

// current state of the aci (setup/standby/active/sleep)

// open remote pipe pending

// close remote pipe pending

// Current pipe available bitmap

// Current pipe closed bitmap

// Current connection interval, slave latency and link supervision timeout

// Current State of the the GATT client (Service Discovery status)

extern hal\_aci\_data\_t msg\_to\_send;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* Utility function to fill the the ACI command queue \*/

/\* aci\_stat Pointer to the ACI state \*/

/\* num\_cmd\_offset(in/out) Offset in the Setup message array to start from \*/

/\* offset is updated to the new index after the queue is filled \*/

/\* or the last message us placed in the queue \*/

/\* Returns true if at least one message was transferred \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

static bool aci\_setup\_fill(aci\_state\_t \*aci\_stat, uint8\_t \*num\_cmd\_offset)

{

 bool ret\_val = false;

 while (\*num\_cmd\_offset < aci\_stat->aci\_setup\_info.num\_setup\_msgs)

 {

 //Board dependent defines

 #if defined (\_\_AVR\_\_)

 //For Arduino copy the setup ACI message from Flash to RAM.

 memcpy\_P(&msg\_to\_send, &(aci\_stat->aci\_setup\_info.setup\_msgs[\*num\_cmd\_offset]),

 pgm\_read\_byte\_near(&(aci\_stat->aci\_setup\_info.setup\_msgs[\*num\_cmd\_offset].buffer[0]))+2);

 #elif defined(\_\_PIC32MX\_\_)

 //In ChipKit we store the setup messages in RAM

 //Add 2 bytes to the length byte for status byte, length for the total number of bytes

 memcpy(&msg\_to\_send, &(aci\_stat->aci\_setup\_info.setup\_msgs[\*num\_cmd\_offset]),

 (aci\_stat->aci\_setup\_info.setup\_msgs[\*num\_cmd\_offset].buffer[0]+2));

 #endif

 //Put the Setup ACI message in the command queue

 if (!hal\_aci\_tl\_send(&msg\_to\_send))

 {

 //ACI Command Queue is full

 // \*num\_cmd\_offset is now pointing to the index of the Setup command that did not get sent

 return ret\_val;

 }

 ret\_val = true;

 (\*num\_cmd\_offset)++;

 }

 return ret\_val;

}

uint8\_t do\_aci\_setup(aci\_state\_t \*aci\_stat)

{

 uint8\_t setup\_offset = 0;

 uint32\_t i = 0x0000;

 aci\_evt\_t \* aci\_evt = NULL;

 aci\_status\_code\_t cmd\_status = ACI\_STATUS\_ERROR\_CRC\_MISMATCH;

 /\*

 We are using the same buffer since we are copying the contents of the buffer

 when queuing and immediately processing the buffer when receiving

 \*/

 hal\_aci\_evt\_t \*aci\_data = (hal\_aci\_evt\_t \*)&msg\_to\_send;

 /\* Messages in the outgoing queue must be handled before the Setup routine can run.

 \* If it is non-empty we return. The user should then process the messages before calling

 \* do\_aci\_setup() again.

 \*/

 if (!lib\_aci\_command\_queue\_empty())

 {

 printf("Error");

 return SETUP\_FAIL\_COMMAND\_QUEUE\_NOT\_EMPTY;

 }

 /\* If there are events pending from the device that are not relevant to setup, we return false

 \* so that the user can handle them. At this point we don't care what the event is,

 \* as any event is an error.

 \*/

 if (lib\_aci\_event\_peek(aci\_data))

 {

 printf("Error2");

 return SETUP\_FAIL\_EVENT\_QUEUE\_NOT\_EMPTY;

 }

 /\* Fill the ACI command queue with as many Setup messages as it will hold. \*/

 aci\_setup\_fill(aci\_stat, &setup\_offset);

 while (cmd\_status != ACI\_STATUS\_TRANSACTION\_COMPLETE)

 {

 /\* This counter is used to ensure that this function does not loop forever. When the device

 \* returns a valid response, we reset the counter.

 \*/

 if (i++ > 0xFFFFE)

 {

 printf("Error3");

 return SETUP\_FAIL\_TIMEOUT;

 }

 if (lib\_aci\_event\_peek(aci\_data))

 {

 aci\_evt = &(aci\_data->evt);

 if (ACI\_EVT\_CMD\_RSP != aci\_evt->evt\_opcode)

 {

 printf("Error4");

 //Receiving something other than a Command Response Event is an error.

 return SETUP\_FAIL\_NOT\_COMMAND\_RESPONSE;

 }

 cmd\_status = (aci\_status\_code\_t) aci\_evt->params.cmd\_rsp.cmd\_status;

 switch (cmd\_status)

 {

 case ACI\_STATUS\_TRANSACTION\_CONTINUE:

 //As the device is responding, reset guard counter

 i = 0;

 /\* As the device has processed the Setup messages we put in the command queue earlier,

 \* we can proceed to fill the queue with new messages

 \*/

 aci\_setup\_fill(aci\_stat, &setup\_offset);

 break;

 case ACI\_STATUS\_TRANSACTION\_COMPLETE:

 //Break out of the while loop when this status code appears

 break;

 default:

 printf("Error5");

 //An event with any other status code should be handled by the application

 return SETUP\_FAIL\_NOT\_SETUP\_EVENT;

 }

 /\* If we haven't returned at this point, the event was either ACI\_STATUS\_TRANSACTION\_CONTINUE

 \* or ACI\_STATUS\_TRANSACTION\_COMPLETE. We don't need the event itself, so we simply

 \* remove it from the queue.

 \*/

 lib\_aci\_event\_get (aci\_stat, aci\_data);

 }

 }

 return SETUP\_SUCCESS;

}

3. aci\_lib.c

#include "hal\_platform.h"

#include "aci.h"

#include "aci\_cmds.h"

#include "aci\_evts.h"

#include "acilib.h"

#include "aci\_protocol\_defines.h"

#include "acilib\_defs.h"

#include "acilib\_if.h"

#include "acilib\_types.h"

#include <string.h>

#include <stdio.h>

#include <stdbool.h>

void acil\_encode\_cmd\_set\_test\_mode(uint8\_t \*buffer, aci\_cmd\_params\_test\_t \*p\_aci\_cmd\_params\_test)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = 2;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_TEST;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_TEST + OFFSET\_ACI\_CMD\_PARAMS\_TEST\_T\_TEST\_MODE\_CHANGE) = p\_aci\_cmd\_params\_test->test\_mode\_change;

}

void acil\_encode\_cmd\_sleep(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = 1;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SLEEP;

}

void acil\_encode\_cmd\_get\_device\_version(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = 1;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_GET\_DEVICE\_VERSION;

}

void acil\_encode\_cmd\_set\_local\_data(uint8\_t \*buffer, aci\_cmd\_params\_set\_local\_data\_t \*p\_aci\_cmd\_params\_set\_local\_data, uint8\_t data\_size)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_SET\_LOCAL\_DATA\_BASE\_LEN + data\_size;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SET\_LOCAL\_DATA;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SET\_LOCAL\_DATA + OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_T\_TX\_DATA + OFFSET\_ACI\_TX\_DATA\_T\_PIPE\_NUMBER) = p\_aci\_cmd\_params\_set\_local\_data->tx\_data.pipe\_number;

 memcpy(buffer + OFFSET\_ACI\_CMD\_T\_SET\_LOCAL\_DATA + OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_T\_TX\_DATA + OFFSET\_ACI\_TX\_DATA\_T\_ACI\_DATA, &(p\_aci\_cmd\_params\_set\_local\_data->tx\_data.aci\_data[0]), data\_size);

}

void acil\_encode\_cmd\_connect(uint8\_t \*buffer, aci\_cmd\_params\_connect\_t \*p\_aci\_cmd\_params\_connect)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_CONNECT\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_CONNECT;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CONNECT + OFFSET\_ACI\_CMD\_PARAMS\_CONNECT\_T\_TIMEOUT\_MSB) = (uint8\_t)(p\_aci\_cmd\_params\_connect->timeout >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CONNECT + OFFSET\_ACI\_CMD\_PARAMS\_CONNECT\_T\_TIMEOUT\_LSB) = (uint8\_t)(p\_aci\_cmd\_params\_connect->timeout);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CONNECT + OFFSET\_ACI\_CMD\_PARAMS\_CONNECT\_T\_ADV\_INTERVAL\_MSB) = (uint8\_t)(p\_aci\_cmd\_params\_connect->adv\_interval >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CONNECT + OFFSET\_ACI\_CMD\_PARAMS\_CONNECT\_T\_ADV\_INTERVAL\_LSB) = (uint8\_t)(p\_aci\_cmd\_params\_connect->adv\_interval);

}

void acil\_encode\_cmd\_bond(uint8\_t \*buffer, aci\_cmd\_params\_bond\_t \*p\_aci\_cmd\_params\_bond)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_BOND\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_BOND;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_BOND + OFFSET\_ACI\_CMD\_PARAMS\_BOND\_T\_TIMEOUT\_MSB) = (uint8\_t)(p\_aci\_cmd\_params\_bond->timeout >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_BOND + OFFSET\_ACI\_CMD\_PARAMS\_BOND\_T\_TIMEOUT\_LSB) = (uint8\_t)(p\_aci\_cmd\_params\_bond->timeout);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_BOND + OFFSET\_ACI\_CMD\_PARAMS\_BOND\_T\_ADV\_INTERVAL\_MSB) = (uint8\_t)(p\_aci\_cmd\_params\_bond->adv\_interval >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_BOND + OFFSET\_ACI\_CMD\_PARAMS\_BOND\_T\_ADV\_INTERVAL\_LSB) = (uint8\_t)(p\_aci\_cmd\_params\_bond->adv\_interval);

}

void acil\_encode\_cmd\_disconnect(uint8\_t \*buffer, aci\_cmd\_params\_disconnect\_t \*p\_aci\_cmd\_params\_disconnect)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_DISCONNECT\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_DISCONNECT;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_DISCONNECT + OFFSET\_ACI\_CMD\_PARAMS\_DISCONNECT\_T\_REASON) = (uint8\_t)(p\_aci\_cmd\_params\_disconnect->reason);

}

void acil\_encode\_baseband\_reset(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_BASEBAND\_RESET\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_RADIO\_RESET;

}

void acil\_encode\_direct\_connect(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_DIRECT\_CONNECT\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_CONNECT\_DIRECT;

}

void acil\_encode\_cmd\_wakeup(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_WAKEUP\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_WAKEUP;

}

void acil\_encode\_cmd\_set\_radio\_tx\_power(uint8\_t \*buffer, aci\_cmd\_params\_set\_tx\_power\_t \*p\_aci\_cmd\_params\_set\_tx\_power)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_SET\_RADIO\_TX\_POWER\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SET\_TX\_POWER;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SET\_TX\_POWER + OFFSET\_ACI\_CMD\_PARAMS\_SET\_TX\_POWER\_T\_DEVICE\_POWER) = (uint8\_t)p\_aci\_cmd\_params\_set\_tx\_power->device\_power;

}

void acil\_encode\_cmd\_get\_address(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_GET\_DEVICE\_ADDR\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_GET\_DEVICE\_ADDRESS;

}

void acil\_encode\_cmd\_send\_data(uint8\_t \*buffer, aci\_cmd\_params\_send\_data\_t \*p\_aci\_cmd\_params\_send\_data\_t, uint8\_t data\_size)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_SEND\_DATA\_BASE\_LEN + data\_size;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SEND\_DATA;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SEND\_DATA + OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_T\_TX\_DATA + OFFSET\_ACI\_TX\_DATA\_T\_PIPE\_NUMBER) = p\_aci\_cmd\_params\_send\_data\_t->tx\_data.pipe\_number;

 memcpy((buffer + OFFSET\_ACI\_CMD\_T\_SEND\_DATA + OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_T\_TX\_DATA + OFFSET\_ACI\_TX\_DATA\_T\_ACI\_DATA), &(p\_aci\_cmd\_params\_send\_data\_t->tx\_data.aci\_data[0]), data\_size);

}

void acil\_encode\_cmd\_request\_data(uint8\_t \*buffer, aci\_cmd\_params\_request\_data\_t \*p\_aci\_cmd\_params\_request\_data)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_DATA\_REQUEST\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_REQUEST\_DATA;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_REQUEST\_DATA + OFFSET\_ACI\_CMD\_PARAMS\_REQUEST\_DATA\_T\_PIPE\_NUMBER) = p\_aci\_cmd\_params\_request\_data->pipe\_number;

}

void acil\_encode\_cmd\_open\_remote\_pipe(uint8\_t \*buffer, aci\_cmd\_params\_open\_remote\_pipe\_t \*p\_aci\_cmd\_params\_open\_remote\_pipe)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_OPEN\_REMOTE\_PIPE\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_OPEN\_REMOTE\_PIPE;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_OPEN\_REMOTE\_PIPE + OFFSET\_ACI\_CMD\_PARAMS\_OPEN\_REMOTE\_PIPE\_T\_PIPE\_NUMBER) = p\_aci\_cmd\_params\_open\_remote\_pipe->pipe\_number;

}

void acil\_encode\_cmd\_close\_remote\_pipe(uint8\_t \*buffer, aci\_cmd\_params\_close\_remote\_pipe\_t \*p\_aci\_cmd\_params\_close\_remote\_pipe)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_CLOSE\_REMOTE\_PIPE\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_CLOSE\_REMOTE\_PIPE;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CLOSE\_REMOTE\_PIPE + OFFSET\_ACI\_CMD\_PARAMS\_CLOSE\_REMOTE\_PIPE\_T\_PIPE\_NUMBER) = p\_aci\_cmd\_params\_close\_remote\_pipe->pipe\_number;

}

void acil\_encode\_cmd\_echo\_msg(uint8\_t \*buffer, aci\_cmd\_params\_echo\_t \*p\_cmd\_params\_echo, uint8\_t msg\_size)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_ECHO\_MSG\_CMD\_BASE\_LEN + msg\_size;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_ECHO;

 memcpy((buffer + OFFSET\_ACI\_CMD\_T\_ECHO + OFFSET\_ACI\_CMD\_PARAMS\_ECHO\_T\_ECHO\_DATA), &(p\_cmd\_params\_echo->echo\_data[0]), msg\_size);

}

void acil\_encode\_cmd\_battery\_level(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = 1;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_GET\_BATTERY\_LEVEL;

}

void acil\_encode\_cmd\_temparature(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = 1;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_GET\_TEMPERATURE;

}

void acil\_encode\_cmd\_read\_dynamic\_data(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = 1;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_READ\_DYNAMIC\_DATA;

}

void acil\_encode\_cmd\_write\_dynamic\_data(uint8\_t \*buffer, uint8\_t seq\_no, uint8\_t\* dynamic\_data, uint8\_t dynamic\_data\_size)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_WRITE\_DYNAMIC\_DATA\_BASE\_LEN + dynamic\_data\_size;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_WRITE\_DYNAMIC\_DATA;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_WRITE\_DYNAMIC\_DATA + OFFSET\_ACI\_CMD\_PARAMS\_WRITE\_DYNAMIC\_DATA\_T\_SEQ\_NO) = seq\_no;

 memcpy((buffer + OFFSET\_ACI\_CMD\_T\_WRITE\_DYNAMIC\_DATA + OFFSET\_ACI\_CMD\_PARAMS\_WRITE\_DYNAMIC\_DATA\_T\_DYNAMIC\_DATA), dynamic\_data, dynamic\_data\_size);

}

void acil\_encode\_cmd\_change\_timing\_req(uint8\_t \*buffer, aci\_cmd\_params\_change\_timing\_t \*p\_aci\_cmd\_params\_change\_timing)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_CHANGE\_TIMING\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_CHANGE\_TIMING;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING + OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS + OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_MIN\_CONN\_INTERVAL\_MSB) = (uint8\_t)(p\_aci\_cmd\_params\_change\_timing->conn\_params.min\_conn\_interval >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING + OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS + OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_MIN\_CONN\_INTERVAL\_LSB) = (uint8\_t)(p\_aci\_cmd\_params\_change\_timing->conn\_params.min\_conn\_interval);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING + OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS + OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_MAX\_CONN\_INTERVAL\_MSB) = (uint8\_t)(p\_aci\_cmd\_params\_change\_timing->conn\_params.max\_conn\_interval >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING + OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS + OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_MAX\_CONN\_INTERVAL\_LSB) = (uint8\_t)(p\_aci\_cmd\_params\_change\_timing->conn\_params.max\_conn\_interval);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING + OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS + OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_SLAVE\_LATENCY\_MSB ) = (uint8\_t)(p\_aci\_cmd\_params\_change\_timing->conn\_params.slave\_latency >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING + OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS + OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_SLAVE\_LATENCY\_LSB ) = (uint8\_t)(p\_aci\_cmd\_params\_change\_timing->conn\_params.slave\_latency);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING + OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS + OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_TIMEOUT\_MULT\_MSB ) = (uint8\_t)(p\_aci\_cmd\_params\_change\_timing->conn\_params.timeout\_mult >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING + OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS + OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_TIMEOUT\_MULT\_LSB ) = (uint8\_t)(p\_aci\_cmd\_params\_change\_timing->conn\_params.timeout\_mult);

}

void acil\_encode\_cmd\_set\_app\_latency(uint8\_t \*buffer, aci\_cmd\_params\_set\_app\_latency\_t \*p\_aci\_cmd\_params\_set\_app\_latency)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_SET\_APP\_LATENCY\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SET\_APP\_LATENCY;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SET\_APP\_LATENCY + OFFSET\_ACI\_CMD\_PARAMS\_SET\_APP\_LATENCY\_T\_MODE) = (uint8\_t)( p\_aci\_cmd\_params\_set\_app\_latency->mode);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SET\_APP\_LATENCY + OFFSET\_ACI\_CMD\_PARAMS\_SET\_APP\_LATENCY\_T\_LATENCY\_MSB) = (uint8\_t)( p\_aci\_cmd\_params\_set\_app\_latency->latency>>8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SET\_APP\_LATENCY + OFFSET\_ACI\_CMD\_PARAMS\_SET\_APP\_LATENCY\_T\_LATENCY\_LSB) = (uint8\_t)( p\_aci\_cmd\_params\_set\_app\_latency->latency);

}

void acil\_encode\_cmd\_change\_timing\_req\_GAP\_PPCP(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_CHANGE\_TIMING\_LEN\_GAP\_PPCP;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_CHANGE\_TIMING;

}

void acil\_encode\_cmd\_setup(uint8\_t \*buffer, aci\_cmd\_params\_setup\_t \*p\_aci\_cmd\_params\_setup, uint8\_t setup\_data\_size)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = setup\_data\_size + MSG\_SETUP\_CMD\_BASE\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SETUP;

 memcpy((buffer + OFFSET\_ACI\_CMD\_T\_SETUP), &(p\_aci\_cmd\_params\_setup->setup\_data[0]), setup\_data\_size);

}

void acil\_encode\_cmd\_dtm\_cmd(uint8\_t \*buffer, aci\_cmd\_params\_dtm\_cmd\_t \*p\_aci\_cmd\_params\_dtm\_cmd)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_DTM\_CMD;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_DTM\_CMD;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_DTM\_CMD) = p\_aci\_cmd\_params\_dtm\_cmd->cmd\_msb;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_DTM\_CMD + 1) = p\_aci\_cmd\_params\_dtm\_cmd->cmd\_lsb;

}

void acil\_encode\_cmd\_send\_data\_ack(uint8\_t \*buffer, const uint8\_t pipe\_number )

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_ACK\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SEND\_DATA\_ACK;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SEND\_DATA\_ACK + OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_ACK\_T\_PIPE\_NUMBER) = pipe\_number;

}

void acil\_encode\_cmd\_send\_data\_nack(uint8\_t \*buffer, const uint8\_t pipe\_number, const uint8\_t err\_code )

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_NACK\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SEND\_DATA\_NACK;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SEND\_DATA\_NACK + OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_NACK\_T\_PIPE\_NUMBER) = pipe\_number;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SEND\_DATA\_NACK + OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_NACK\_T\_ERROR\_CODE) = err\_code;

}

void acil\_encode\_cmd\_bond\_security\_request(uint8\_t \*buffer)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = 1;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_BOND\_SECURITY\_REQUEST;

}

void acil\_encode\_cmd\_broadcast(uint8\_t \*buffer, aci\_cmd\_params\_broadcast\_t \* p\_aci\_cmd\_params\_broadcast)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_BROADCAST\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_BROADCAST;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_BROADCAST + OFFSET\_ACI\_CMD\_PARAMS\_BROADCAST\_T\_TIMEOUT\_LSB) = (p\_aci\_cmd\_params\_broadcast->timeout & 0xff);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_BROADCAST + OFFSET\_ACI\_CMD\_PARAMS\_BROADCAST\_T\_TIMEOUT\_MSB) = (uint8\_t)(p\_aci\_cmd\_params\_broadcast->timeout >> 8);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_BROADCAST + OFFSET\_ACI\_CMD\_PARAMS\_BROADCAST\_T\_ADV\_INTERVAL\_LSB) = (p\_aci\_cmd\_params\_broadcast->adv\_interval & 0xff);

 \*(buffer + OFFSET\_ACI\_CMD\_T\_BROADCAST + OFFSET\_ACI\_CMD\_PARAMS\_BROADCAST\_T\_ADV\_INTERVAL\_MSB) = (uint8\_t)(p\_aci\_cmd\_params\_broadcast->adv\_interval >> 8);

}

void acil\_encode\_cmd\_open\_adv\_pipes(uint8\_t \*buffer, aci\_cmd\_params\_open\_adv\_pipe\_t \* p\_aci\_cmd\_params\_open\_adv\_pipe)

{

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = MSG\_OPEN\_ADV\_PIPES\_LEN;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_OPEN\_ADV\_PIPE;

 memcpy(buffer + OFFSET\_ACI\_CMD\_T\_OPEN\_ADV\_PIPE + OFFSET\_ACI\_CMD\_PARAMS\_OPEN\_ADV\_PIPE\_T\_PIPES, p\_aci\_cmd\_params\_open\_adv\_pipe->pipes, 8);

}

void acil\_encode\_cmd\_set\_key(uint8\_t \*buffer, aci\_cmd\_params\_set\_key\_t \*p\_aci\_cmd\_params\_set\_key)

{

 /\*

 The length of the key is computed based on the type of key transaction.

 - Key Reject

 - Key type is passkey

 \*/

 uint8\_t len;

 switch (p\_aci\_cmd\_params\_set\_key->key\_type)

 {

 case ACI\_KEY\_TYPE\_INVALID:

 len = MSG\_SET\_KEY\_REJECT\_LEN;

 break;

 case ACI\_KEY\_TYPE\_PASSKEY:

 len = MSG\_SET\_KEY\_PASSKEY\_LEN;

 break;

 default:

 len=0;

 break;

 }

 \*(buffer + OFFSET\_ACI\_CMD\_T\_LEN) = len;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE) = ACI\_CMD\_SET\_KEY;

 \*(buffer + OFFSET\_ACI\_CMD\_T\_SET\_KEY + OFFSET\_ACI\_CMD\_PARAMS\_SET\_KEY\_T\_KEY\_TYPE) = p\_aci\_cmd\_params\_set\_key->key\_type;

 memcpy((buffer + OFFSET\_ACI\_CMD\_T\_SET\_KEY + OFFSET\_ACI\_CMD\_PARAMS\_SET\_KEY\_T\_PASSKEY), (uint8\_t \* )&(p\_aci\_cmd\_params\_set\_key->key), len-2);//Reducing 2 for the opcode byte and type

}

bool acil\_encode\_cmd(uint8\_t \*buffer, aci\_cmd\_t \*p\_aci\_cmd)

{

 bool ret\_val = false;

 switch(p\_aci\_cmd->cmd\_opcode)

 {

 case ACI\_CMD\_TEST:

 acil\_encode\_cmd\_set\_test\_mode(buffer, &(p\_aci\_cmd->params.test));

 break;

 case ACI\_CMD\_SLEEP:

 acil\_encode\_cmd\_sleep(buffer);

 break;

 case ACI\_CMD\_GET\_DEVICE\_VERSION:

 acil\_encode\_cmd\_get\_device\_version(buffer);

 break;

 case ACI\_CMD\_WAKEUP:

 acil\_encode\_cmd\_wakeup(buffer);

 break;

 case ACI\_CMD\_ECHO:

 acil\_encode\_cmd\_echo\_msg(buffer, &(p\_aci\_cmd->params.echo), (p\_aci\_cmd->len - MSG\_ECHO\_MSG\_CMD\_BASE\_LEN));

 break;

 case ACI\_CMD\_GET\_BATTERY\_LEVEL:

 acil\_encode\_cmd\_battery\_level(buffer);

 break;

 case ACI\_CMD\_GET\_TEMPERATURE:

 acil\_encode\_cmd\_temparature(buffer);

 break;

 case ACI\_CMD\_GET\_DEVICE\_ADDRESS:

 acil\_encode\_cmd\_get\_address(buffer);

 break;

 case ACI\_CMD\_SET\_TX\_POWER:

 acil\_encode\_cmd\_set\_radio\_tx\_power(buffer, &(p\_aci\_cmd->params.set\_tx\_power));

 break;

 case ACI\_CMD\_CONNECT:

 acil\_encode\_cmd\_connect(buffer, &(p\_aci\_cmd->params.connect));

 break;

 case ACI\_CMD\_BOND:

 acil\_encode\_cmd\_bond(buffer, &(p\_aci\_cmd->params.bond));

 break;

 case ACI\_CMD\_DISCONNECT:

 acil\_encode\_cmd\_disconnect(buffer, &(p\_aci\_cmd->params.disconnect));

 break;

 case ACI\_CMD\_RADIO\_RESET:

 acil\_encode\_baseband\_reset(buffer);

 break;

 case ACI\_CMD\_CHANGE\_TIMING:

 acil\_encode\_cmd\_change\_timing\_req(buffer, &(p\_aci\_cmd->params.change\_timing));

 break;

 case ACI\_CMD\_SETUP:

 acil\_encode\_cmd\_setup(buffer, &(p\_aci\_cmd->params.setup), (p\_aci\_cmd->len - MSG\_SETUP\_CMD\_BASE\_LEN));

 break;

 case ACI\_CMD\_DTM\_CMD:

 acil\_encode\_cmd\_dtm\_cmd(buffer, &(p\_aci\_cmd->params.dtm\_cmd));

 break;

 case ACI\_CMD\_READ\_DYNAMIC\_DATA:

 acil\_encode\_cmd\_read\_dynamic\_data(buffer);

 break;

 case ACI\_CMD\_WRITE\_DYNAMIC\_DATA:

 acil\_encode\_cmd\_write\_dynamic\_data(buffer, p\_aci\_cmd->params.write\_dynamic\_data.seq\_no, &(p\_aci\_cmd->params.write\_dynamic\_data.dynamic\_data[0]), (p\_aci\_cmd->len - MSG\_WRITE\_DYNAMIC\_DATA\_BASE\_LEN));

 break;

 case ACI\_CMD\_OPEN\_REMOTE\_PIPE:

 acil\_encode\_cmd\_open\_remote\_pipe(buffer, &(p\_aci\_cmd->params.open\_remote\_pipe));

 break;

 case ACI\_CMD\_SEND\_DATA:

 acil\_encode\_cmd\_send\_data(buffer, &(p\_aci\_cmd->params.send\_data), (p\_aci\_cmd->len - MSG\_SEND\_DATA\_BASE\_LEN));

 break;

 case ACI\_CMD\_SEND\_DATA\_ACK:

 acil\_encode\_cmd\_send\_data\_ack(buffer, p\_aci\_cmd->params.send\_data\_ack.pipe\_number );

 break;

 case ACI\_CMD\_REQUEST\_DATA:

 acil\_encode\_cmd\_request\_data(buffer, &(p\_aci\_cmd->params.request\_data));

 break;

 case ACI\_CMD\_SET\_LOCAL\_DATA:

 acil\_encode\_cmd\_set\_local\_data(buffer, (aci\_cmd\_params\_set\_local\_data\_t \*)(&(p\_aci\_cmd->params.send\_data)), (p\_aci\_cmd->len - MSG\_SET\_LOCAL\_DATA\_BASE\_LEN));

 break;

 case ACI\_CMD\_BOND\_SECURITY\_REQUEST:

 acil\_encode\_cmd\_bond\_security\_request(buffer);

 break;

 default:

 break;

 }

 return ret\_val;

}

void acil\_decode\_evt\_command\_response(uint8\_t \*buffer\_in, aci\_evt\_params\_cmd\_rsp\_t \*p\_evt\_params\_cmd\_rsp)

{

 aci\_evt\_cmd\_rsp\_params\_get\_device\_version\_t \*p\_device\_version;

 aci\_evt\_cmd\_rsp\_params\_get\_device\_address\_t \*p\_device\_address;

 aci\_evt\_cmd\_rsp\_params\_get\_temperature\_t \*p\_temperature;

 aci\_evt\_cmd\_rsp\_params\_get\_battery\_level\_t \*p\_batt\_lvl;

 aci\_evt\_cmd\_rsp\_read\_dynamic\_data\_t \*p\_read\_dyn\_data;

 aci\_evt\_cmd\_rsp\_params\_dtm\_cmd\_t \*p\_dtm\_evt;

 p\_evt\_params\_cmd\_rsp->cmd\_opcode = (aci\_cmd\_opcode\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_CMD\_OPCODE);

 p\_evt\_params\_cmd\_rsp->cmd\_status = (aci\_status\_code\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_CMD\_STATUS);

 switch (p\_evt\_params\_cmd\_rsp->cmd\_opcode)

 {

 case ACI\_CMD\_GET\_DEVICE\_VERSION:

 p\_device\_version = &(p\_evt\_params\_cmd\_rsp->params.get\_device\_version);

 p\_device\_version->configuration\_id = (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_CONFIGURATION\_ID\_LSB);

 p\_device\_version->configuration\_id |= (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_CONFIGURATION\_ID\_MSB) << 8;

 p\_device\_version->aci\_version = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_ACI\_VERSION);

 p\_device\_version->setup\_format = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_FORMAT);

 p\_device\_version->setup\_id = (uint32\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_ID\_LSB0);

 p\_device\_version->setup\_id |= (uint32\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_ID\_LSB1) << 8;

 p\_device\_version->setup\_id |= (uint32\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_ID\_MSB0) << 16;

 p\_device\_version->setup\_id |= (uint32\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_ID\_MSB1) << 24;

 p\_device\_version->setup\_status = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_STATUS);

 break;

 case ACI\_CMD\_GET\_DEVICE\_ADDRESS:

 p\_device\_address = &(p\_evt\_params\_cmd\_rsp->params.get\_device\_address);

 memcpy((uint8\_t \*)(p\_device\_address->bd\_addr\_own), (buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP+OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_ADDRESS+OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_ADDRESS\_T\_BD\_ADDR\_OWN), BTLE\_DEVICE\_ADDRESS\_SIZE);

 p\_device\_address->bd\_addr\_type = (aci\_bd\_addr\_type\_t) \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP+OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_ADDRESS+OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_ADDRESS\_T\_BD\_ADDR\_TYPE);

 break;

 case ACI\_CMD\_GET\_TEMPERATURE:

 p\_temperature = &(p\_evt\_params\_cmd\_rsp->params.get\_temperature);

 p\_temperature->temperature\_value = (int16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_TEMPERATURE + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_TEMPERATURE\_T\_TEMPERATURE\_VALUE\_LSB);

 p\_temperature->temperature\_value |= (int16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_TEMPERATURE + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_TEMPERATURE\_T\_TEMPERATURE\_VALUE\_MSB) << 8;

 break;

 case ACI\_CMD\_GET\_BATTERY\_LEVEL:

 p\_batt\_lvl = &(p\_evt\_params\_cmd\_rsp->params.get\_battery\_level);

 p\_batt\_lvl->battery\_level = (int16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_BATTERY\_LEVEL + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_BATTERY\_LEVEL\_T\_BATTERY\_LEVEL\_LSB);

 p\_batt\_lvl->battery\_level |= (int16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_BATTERY\_LEVEL + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_BATTERY\_LEVEL\_T\_BATTERY\_LEVEL\_MSB) << 8;

 break;

 case ACI\_CMD\_READ\_DYNAMIC\_DATA:

 p\_read\_dyn\_data = &(p\_evt\_params\_cmd\_rsp->params.read\_dynamic\_data);

 p\_read\_dyn\_data->seq\_no = (uint8\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_READ\_DYNAMIC\_DATA + OFFSET\_ACI\_EVT\_CMD\_RSP\_READ\_DYNAMIC\_DATA\_T\_SEQ\_NO);

 memcpy((uint8\_t \*)(p\_read\_dyn\_data->dynamic\_data), (buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_READ\_DYNAMIC\_DATA + OFFSET\_ACI\_CMD\_PARAMS\_WRITE\_DYNAMIC\_DATA\_T\_DYNAMIC\_DATA), ACIL\_DECODE\_EVT\_GET\_LENGTH(buffer\_in) - 3); // 3 bytes subtracted account for EventCode, CommandOpCode and Status bytes.

 // Now that the p\_read\_dyn\_data->dynamic\_data will be pointing to memory location with enough space to accommodate upto 27 bytes of dynamic data received. This is because of the padding element in aci\_evt\_params\_cmd\_rsp\_t

 break;

 case ACI\_CMD\_DTM\_CMD:

 p\_dtm\_evt = &(p\_evt\_params\_cmd\_rsp->params.dtm\_cmd);

 p\_dtm\_evt->evt\_msb = (uint8\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_DTM\_CMD + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_DTM\_CMD\_T\_EVT\_MSB);

 p\_dtm\_evt->evt\_lsb = (uint8\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CMD\_RSP + OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_DTM\_CMD + OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_DTM\_CMD\_T\_EVT\_LSB);

 break;

 }

}

void acil\_decode\_evt\_device\_started(uint8\_t \*buffer\_in, aci\_evt\_params\_device\_started\_t \*p\_evt\_params\_device\_started)

{

 p\_evt\_params\_device\_started->device\_mode = (aci\_device\_operation\_mode\_t) \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DEVICE\_STARTED+OFFSET\_ACI\_EVT\_PARAMS\_DEVICE\_STARTED\_T\_DEVICE\_MODE);

 p\_evt\_params\_device\_started->hw\_error = (aci\_hw\_error\_t) \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DEVICE\_STARTED+OFFSET\_ACI\_EVT\_PARAMS\_DEVICE\_STARTED\_T\_HW\_ERROR);

 p\_evt\_params\_device\_started->credit\_available = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DEVICE\_STARTED+OFFSET\_ACI\_EVT\_PARAMS\_DEVICE\_STARTED\_T\_CREDIT\_AVAILABLE);

}

void acil\_decode\_evt\_pipe\_status(uint8\_t \*buffer\_in, aci\_evt\_params\_pipe\_status\_t \*p\_aci\_evt\_params\_pipe\_status)

{

 memcpy((uint8\_t \*)p\_aci\_evt\_params\_pipe\_status->pipes\_open\_bitmap, (buffer\_in + OFFSET\_ACI\_EVT\_T\_PIPE\_STATUS + OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_STATUS\_T\_PIPES\_OPEN\_BITMAP), 8);

 memcpy((uint8\_t \*)p\_aci\_evt\_params\_pipe\_status->pipes\_closed\_bitmap, (buffer\_in + OFFSET\_ACI\_EVT\_T\_PIPE\_STATUS + OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_STATUS\_T\_PIPES\_CLOSED\_BITMAP), 8);

}

void acil\_decode\_evt\_disconnected(uint8\_t \*buffer\_in, aci\_evt\_params\_disconnected\_t \*p\_aci\_evt\_params\_disconnected)

{

 p\_aci\_evt\_params\_disconnected->aci\_status = (aci\_status\_code\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DISCONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_DISCONNECTED\_T\_ACI\_STATUS);

 p\_aci\_evt\_params\_disconnected->btle\_status = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DISCONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_DISCONNECTED\_T\_BTLE\_STATUS);

}

void acil\_decode\_evt\_bond\_status(uint8\_t \*buffer\_in, aci\_evt\_params\_bond\_status\_t \*p\_aci\_evt\_params\_bond\_status)

{

 p\_aci\_evt\_params\_bond\_status->status\_code = (aci\_bond\_status\_code\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_BOND\_STATUS + OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_STATUS\_CODE);

 p\_aci\_evt\_params\_bond\_status->status\_source = (aci\_bond\_status\_source\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_BOND\_STATUS + OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_STATUS\_SOURCE);

 p\_aci\_evt\_params\_bond\_status->secmode1\_bitmap = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_BOND\_STATUS + OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_SECMODE1\_BITMAP);

 p\_aci\_evt\_params\_bond\_status->secmode2\_bitmap = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_BOND\_STATUS + OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_SECMODE2\_BITMAP);

 p\_aci\_evt\_params\_bond\_status->keys\_exchanged\_slave = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_BOND\_STATUS + OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_KEYS\_EXCHANGED\_SLAVE);

 p\_aci\_evt\_params\_bond\_status->keys\_exchanged\_master = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_BOND\_STATUS + OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_KEYS\_EXCHANGED\_MASTER);

}

uint8\_t acil\_decode\_evt\_data\_received(uint8\_t \*buffer\_in, aci\_evt\_params\_data\_received\_t \*p\_evt\_params\_data\_received)

{

 uint8\_t size = \*( buffer\_in + OFFSET\_ACI\_EVT\_T\_LEN) - (OFFSET\_ACI\_EVT\_T\_DATA\_RECEIVED + OFFSET\_ACI\_RX\_DATA\_T\_ACI\_DATA) + 1 ;

 p\_evt\_params\_data\_received->rx\_data.pipe\_number = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DATA\_RECEIVED + OFFSET\_ACI\_RX\_DATA\_T\_PIPE\_NUMBER);

 memcpy((uint8\_t \*)p\_evt\_params\_data\_received->rx\_data.aci\_data, (buffer\_in + OFFSET\_ACI\_EVT\_T\_DATA\_RECEIVED + OFFSET\_ACI\_RX\_DATA\_T\_ACI\_DATA), size);

 return size;

}

void acil\_decode\_evt\_data\_ack(uint8\_t \*buffer\_in, aci\_evt\_params\_data\_ack\_t \*p\_evt\_params\_data\_ack)

{

 p\_evt\_params\_data\_ack->pipe\_number = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DATA\_ACK + OFFSET\_ACI\_EVT\_PARAMS\_DATA\_ACK\_T\_PIPE\_NUMBER);

}

uint8\_t acil\_decode\_evt\_hw\_error(uint8\_t \*buffer\_in, aci\_evt\_params\_hw\_error\_t \*p\_aci\_evt\_params\_hw\_error)

{

 uint8\_t size = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_LEN) - (OFFSET\_ACI\_EVT\_T\_HW\_ERROR + OFFSET\_ACI\_EVT\_PARAMS\_HW\_ERROR\_T\_FILE\_NAME) + 1;

 p\_aci\_evt\_params\_hw\_error->line\_num = (uint16\_t)(\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_HW\_ERROR + OFFSET\_ACI\_EVT\_PARAMS\_HW\_ERROR\_T\_LINE\_NUM\_MSB)) << 8;

 p\_aci\_evt\_params\_hw\_error->line\_num |= (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_HW\_ERROR + OFFSET\_ACI\_EVT\_PARAMS\_HW\_ERROR\_T\_LINE\_NUM\_LSB);

 memcpy((uint8\_t \*)p\_aci\_evt\_params\_hw\_error->file\_name, (buffer\_in + OFFSET\_ACI\_EVT\_T\_HW\_ERROR + OFFSET\_ACI\_EVT\_PARAMS\_HW\_ERROR\_T\_FILE\_NAME), size);

 return size;

}

void acil\_decode\_evt\_credit(uint8\_t \*buffer\_in, aci\_evt\_params\_data\_credit\_t \*p\_evt\_params\_data\_credit)

{

 p\_evt\_params\_data\_credit->credit = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DATA\_CREDIT + OFFSET\_ACI\_EVT\_PARAMS\_DATA\_CREDIT\_T\_CREDIT);

}

void acil\_decode\_evt\_connected(uint8\_t \*buffer\_in, aci\_evt\_params\_connected\_t \*p\_aci\_evt\_params\_connected)

{

 p\_aci\_evt\_params\_connected->dev\_addr\_type = (aci\_bd\_addr\_type\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_DEV\_ADDR\_TYPE);

 memcpy(&(p\_aci\_evt\_params\_connected->dev\_addr[0]), (buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_DEV\_ADDR), BTLE\_DEVICE\_ADDRESS\_SIZE);

 p\_aci\_evt\_params\_connected->conn\_rf\_interval = (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_RF\_INTERVAL\_MSB) << 8;

 p\_aci\_evt\_params\_connected->conn\_rf\_interval |= (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_RF\_INTERVAL\_LSB);

 p\_aci\_evt\_params\_connected->conn\_slave\_rf\_latency = (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_SLAVE\_RF\_LATENCY\_MSB) << 8;

 p\_aci\_evt\_params\_connected->conn\_slave\_rf\_latency |= (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_SLAVE\_RF\_LATENCY\_LSB);

 p\_aci\_evt\_params\_connected->conn\_rf\_timeout = (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_RF\_TIMEOUT\_MSB) << 8;

 p\_aci\_evt\_params\_connected->conn\_rf\_timeout |= (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_RF\_TIMEOUT\_LSB);

 p\_aci\_evt\_params\_connected->master\_clock\_accuracy = (aci\_clock\_accuracy\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_CONNECTED + OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_MASTER\_CLOCK\_ACCURACY);

}

void acil\_decode\_evt\_timing(uint8\_t \*buffer\_in, aci\_evt\_params\_timing\_t \*p\_evt\_params\_timing)

{

 p\_evt\_params\_timing->conn\_rf\_interval = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_TIMING + OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_RF\_INTERVAL\_MSB) << 8;

 p\_evt\_params\_timing->conn\_rf\_interval |= (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_TIMING + OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_RF\_INTERVAL\_LSB);

 p\_evt\_params\_timing->conn\_slave\_rf\_latency = (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_TIMING + OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_SLAVE\_RF\_LATENCY\_MSB) << 8;

 p\_evt\_params\_timing->conn\_slave\_rf\_latency |= (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_TIMING + OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_SLAVE\_RF\_LATENCY\_LSB);

 p\_evt\_params\_timing->conn\_rf\_timeout = (uint16\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_TIMING + OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_RF\_TIMEOUT\_MSB) << 8;

 p\_evt\_params\_timing->conn\_rf\_timeout |= \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_TIMING + OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_RF\_TIMEOUT\_LSB);

}

void acil\_decode\_evt\_pipe\_error(uint8\_t \*buffer\_in, aci\_evt\_params\_pipe\_error\_t \*p\_evt\_params\_pipe\_error)

{

 //volatile uint8\_t size = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_LEN) - (OFFSET\_ACI\_EVT\_T\_PIPE\_ERROR + OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_ERROR\_T\_ERROR\_DATA) + 1;

 p\_evt\_params\_pipe\_error->pipe\_number = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_PIPE\_ERROR + OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_ERROR\_T\_PIPE\_NUMBER);

 p\_evt\_params\_pipe\_error->error\_code = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_PIPE\_ERROR + OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_ERROR\_T\_ERROR\_CODE);

 p\_evt\_params\_pipe\_error->params.error\_data.content[0] = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_PIPE\_ERROR + OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_ERROR\_T\_ERROR\_DATA + OFFSET\_ERROR\_DATA\_T\_CONTENT);

}

void acil\_decode\_evt\_key\_request(uint8\_t \*buffer\_in, aci\_evt\_params\_key\_request\_t \*p\_evt\_params\_key\_request)

{

 p\_evt\_params\_key\_request->key\_type = (aci\_key\_type\_t)\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_KEY\_REQUEST + OFFSET\_ACI\_EVT\_PARAMS\_KEY\_REQUEST\_T\_KEY\_TYPE);

}

uint8\_t acil\_decode\_evt\_echo(uint8\_t \*buffer\_in, aci\_evt\_params\_echo\_t \*aci\_evt\_params\_echo)

{

 uint8\_t size = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_LEN) - 1;

 memcpy(&aci\_evt\_params\_echo->echo\_data[0], (buffer\_in + OFFSET\_ACI\_EVT\_T\_EVT\_OPCODE +1), size);

 return size;

}

void acil\_decode\_evt\_display\_passkey(uint8\_t \*buffer\_in, aci\_evt\_params\_display\_passkey\_t \*p\_aci\_evt\_params\_display\_passkey)

{

 p\_aci\_evt\_params\_display\_passkey->passkey[0] = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DISPLAY\_PASSKEY + OFFSET\_ACI\_EVT\_PARAMS\_DISPLAY\_PASSKEY\_T\_PASSKEY + 0);

 p\_aci\_evt\_params\_display\_passkey->passkey[1] = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DISPLAY\_PASSKEY + OFFSET\_ACI\_EVT\_PARAMS\_DISPLAY\_PASSKEY\_T\_PASSKEY + 1);

 p\_aci\_evt\_params\_display\_passkey->passkey[2] = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DISPLAY\_PASSKEY + OFFSET\_ACI\_EVT\_PARAMS\_DISPLAY\_PASSKEY\_T\_PASSKEY + 2);

 p\_aci\_evt\_params\_display\_passkey->passkey[3] = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DISPLAY\_PASSKEY + OFFSET\_ACI\_EVT\_PARAMS\_DISPLAY\_PASSKEY\_T\_PASSKEY + 3);

 p\_aci\_evt\_params\_display\_passkey->passkey[4] = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DISPLAY\_PASSKEY + OFFSET\_ACI\_EVT\_PARAMS\_DISPLAY\_PASSKEY\_T\_PASSKEY + 4);

 p\_aci\_evt\_params\_display\_passkey->passkey[5] = \*(buffer\_in + OFFSET\_ACI\_EVT\_T\_DISPLAY\_PASSKEY + OFFSET\_ACI\_EVT\_PARAMS\_DISPLAY\_PASSKEY\_T\_PASSKEY + 5);

}

bool acil\_decode\_evt(uint8\_t \*buffer\_in, aci\_evt\_t \*p\_aci\_evt)

{

 bool ret\_val = true;

 p\_aci\_evt->len = ACIL\_DECODE\_EVT\_GET\_LENGTH(buffer\_in);

 printf("%d", buffer\_in);

 printf("HelloWorld");

 p\_aci\_evt->evt\_opcode = (aci\_evt\_opcode\_t)ACIL\_DECODE\_EVT\_GET\_OPCODE(buffer\_in);

 switch(p\_aci\_evt->evt\_opcode)

 {

 case ACI\_EVT\_DEVICE\_STARTED:

 acil\_decode\_evt\_device\_started(buffer\_in, &(p\_aci\_evt->params.device\_started));

 break;

 case ACI\_EVT\_HW\_ERROR:

 acil\_decode\_evt\_hw\_error(buffer\_in, &(p\_aci\_evt->params.hw\_error));

 break;

 case ACI\_EVT\_CMD\_RSP:

 acil\_decode\_evt\_command\_response(buffer\_in, &(p\_aci\_evt->params.cmd\_rsp));

 break;

 case ACI\_EVT\_DATA\_CREDIT:

 acil\_decode\_evt\_credit(buffer\_in, &(p\_aci\_evt->params.data\_credit));

 break;

 case ACI\_EVT\_CONNECTED:

 acil\_decode\_evt\_connected(buffer\_in, &(p\_aci\_evt->params.connected));

 break;

 case ACI\_EVT\_PIPE\_STATUS:

 acil\_decode\_evt\_pipe\_status(buffer\_in, &(p\_aci\_evt->params.pipe\_status));

 break;

 case ACI\_EVT\_DISCONNECTED:

 acil\_decode\_evt\_disconnected(buffer\_in, &(p\_aci\_evt->params.disconnected));

 break;

 case ACI\_EVT\_BOND\_STATUS:

 acil\_decode\_evt\_bond\_status(buffer\_in, &(p\_aci\_evt->params.bond\_status));

 break;

 case ACI\_EVT\_TIMING:

 acil\_decode\_evt\_timing(buffer\_in, &(p\_aci\_evt->params.timing));

 break;

 case ACI\_EVT\_DATA\_ACK:

 acil\_decode\_evt\_data\_ack(buffer\_in, &(p\_aci\_evt->params.data\_ack));

 break;

 case ACI\_EVT\_DATA\_RECEIVED:

 acil\_decode\_evt\_data\_received(buffer\_in, &(p\_aci\_evt->params.data\_received));

 break;

 case ACI\_EVT\_PIPE\_ERROR:

 acil\_decode\_evt\_pipe\_error(buffer\_in, &(p\_aci\_evt->params.pipe\_error));

 break;

 case ACI\_EVT\_KEY\_REQUEST:

 acil\_decode\_evt\_key\_request(buffer\_in, &(p\_aci\_evt->params.key\_request));

 break;

 case ACI\_EVT\_DISPLAY\_PASSKEY:

 acil\_decode\_evt\_display\_passkey(buffer\_in, &(p\_aci\_evt->params.display\_passkey));

 break;

 default:

 ret\_val = false;

 break;

 }

 return ret\_val;

}

4. ble\_mainLOOP.c

#include <stdio.h>

#include <stdlib.h>

//#include "bleconfigbitsrev2014vC.h"

//#include "SDlib.h"

//#include "SDlib\_delay.h"

//#include <xc.h>

//#include "globalvariables.h" // this is the one that contains the flash and SPI4 functions

#include "sleepmodeHEADER.h"

#include "ADHEADER.h"

#include "newtooth2.h"

//#include "globalvariables.h"

//#include <sys/attribs.h>

//#include <stdbool.h>

 #define SAMPLES 24

 #define DISREGARD 3

int main(int argc, char\*\* argv) {

 /\*\*\*\*Initialization Functions\*\*\*\*\*/

 setup(); // bluetooth setup I think this also initiates the SPI

 ADC\_init(); // A/D initialization

 // set up the flash

 //releasebitprotection();

 //chipERASE();

 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

 int y=0;

 //while(y<5)

 while(1)

 {

 ///Just for testing blink the lights

 TRISE &= 0xFF00; //set Least Significant Byte (last 8 bits) as output

 LATEbits.LATE0 = 1;

 LATEbits.LATE1 = 1;

 LATEbits.LATE2 = 1;

 LATEbits.LATE3 = 1;

 LATEbits.LATE4 = 1;

 LATEbits.LATE5 = 1;

 LATEbits.LATE6 = 1;

 LATEbits.LATE7 = 1;

 LATEbits.LATE1=0;

 delay\_ms(200);

 LATEbits.LATE1=1;

 // declare the variables needed

 float dig;

 float dig1;

 unsigned int dig2;

 unsigned int count = 0;

 unsigned int oneMINUTE[SAMPLES];

 //while loop for continuous sampling (uneeded when using sleep mode

 //while(1)

 int k=0;

 for(k=0; k<SAMPLES; k++)

 {

 dig1 = ADC();

 dig = (dig1\*3.300/1023); //scale

 /\*should be \*100 but had to change to \*10 for the exmaple i was working on\*/

 dig2=(float)dig\*100;

 //dig2=(float)dig\*10;

 //dig2=2\*k;

 oneMINUTE[k]= dig2;

 //check\_address();

 //byte\_write(dig2);

 sleep(); //looking at it on the logic analyzer so commented this out for now

 }

 int i=0;

 int j =i+1;

 unsigned int temp;

 for (i=0; i<SAMPLES; i++)

 //while(i<SAMPLES)

 {

 //while(j<SAMPLES)

 for (j=i+1; j<SAMPLES; j++)

 {

 if(oneMINUTE[i] > oneMINUTE[j])

 {

 temp = oneMINUTE[i];

 oneMINUTE[i] = oneMINUTE[j];

 oneMINUTE[j] = temp;

 } // end if

 //j++;

 } // end for (j)

 // i++;

 } // end for (i)

 unsigned int sum=0;

 unsigned int ave = 0;

 sum = oneMINUTE[SAMPLES-(DISREGARD)-1]+oneMINUTE[SAMPLES-(DISREGARD)-2]+oneMINUTE[SAMPLES-(DISREGARD)-3]+oneMINUTE[SAMPLES-(DISREGARD)-4]+oneMINUTE[SAMPLES-(DISREGARD)-5];

 ave = sum/5;

 // Just doing this until we can get the bluetooth worked out

 //check\_address();

 //byte\_write(ave);

 //readALL();

 LATEbits.LATE1=0;

 delay\_ms(200);

 LATEbits.LATE1=1;

/\*\*\*\*ALL THE FLASH FUNCTIONS HAVE BEEN COMMENTED OUT\*\*\*\*\*/

/\*This is because on the kit board we can't have flash and Bluetooth connected

 at the same time. On the new board we need to uncomment out all of these and

 all the stuff inside the "if(count > 0)" statement\*/

count = 0;

 float data2 = 4.412;

// Attempt Bluetooth Transfer

bool connect = ble\_connect(ave);

if (connect)

{

 if (count == 0)

 {

 ble\_transfer(ave);

 //chipERASE();

 } // end if(count)

 if(count > 0)

 {

// LATBbits.LATB8= 0;

// unsigned int writeEN = SPItransfer (0x06);

// LATBbits.LATB8 = 1;

//

// LATBbits.LATB8 = 0;

// unsigned int readIN = SPItransfer(0x0B);

// readIN = SPItransfer(0x00);

// readIN = SPItransfer(0x00);

// readIN = SPItransfer(0x00);

//

// readIN = SPItransfer(0x00);

// unsigned int data = SPItransfer (0x00);

//

// while (data != 0xFF)

// {

// if (connect == 0) // need to keep checking if connected

// {

// check\_address();

// byte\_write(ave);

// count++;

// break; // should break out of the while loop??

// }

//

// if (connect != 0)// need to keep checking if connected

// {

// ble\_transfer(ave);

// data = SPItransfer(0x00);

// }

// } // end while

// LATBbits.LATB8 = 1;

 if (connect != 0)// need to keep checking if connected

 {

 ble\_transfer(ave);

 //chipERASE();

 }

 } // end it (count>0)

} // end if connect

if(!connect)

{

 //check\_address();

 //byte\_write(ave);

 count++;

} // end if(!connect)

y++;

 } // end while (y<5)

 return (EXIT\_SUCCESS);

 }

5. hal\_aci\_tl.c

#include "hal\_platform.h"

#include "hal\_aci\_tl.h"

#include "aci\_queue.h"

#include <xc.h>

#include <stdbool.h>

//#include "configbitsrev2014vC.h"

#if ( !defined(\_\_SAM3X8E\_\_) && !defined(\_\_PIC32MX\_\_) )

#include <avr/sleep.h>

#endif

/\*

PIC32 supports only MSbit transfer on SPI and the nRF8001 uses LSBit

Use the REVERSE\_BITS macro to convert from MSBit to LSBit

The outgoing command and the incoming event needs to be converted

\*/

//Board dependent defines

//#if defined (\_\_AVR\_\_)

 //For Arduino add nothing

//#elif defined(\_\_PIC32MX\_\_)

 //For ChipKit as the transmission has to be reversed, the next definitions have to be added

 #define REVERSE\_BITS(byte) (((reverse\_lookup[(byte & 0x0F)]) << 4) + reverse\_lookup[((byte & 0xF0) >> 4)])

 static const uint8\_t reverse\_lookup[] = { 0, 8, 4, 12, 2, 10, 6, 14,1, 9, 5, 13,3, 11, 7, 15 };

//#endif

void SPIbegin(void);

void setBitOrder (); // needs to be completed

uint8\_t SPIsend(uint8\_t reverse\_aci\_byte);

void SPIwrite(uint8\_t datatowrite);

uint8\_t SPIread(void);

uint8\_t SPItransfer(uint8\_t reverse\_aci\_byte);

static void m\_aci\_data\_print(hal\_aci\_data\_t \*p\_data);

static void m\_aci\_event\_check(void);

//static void m\_aci\_isr(void);

static void m\_aci\_pins\_set(aci\_pins\_t \*a\_pins\_ptr);

static inline void m\_aci\_reqn\_disable (void);

static inline void m\_aci\_reqn\_enable (void);

static void m\_aci\_q\_flush(void);

static bool m\_aci\_spi\_transfer(hal\_aci\_data\_t \* data\_to\_send, hal\_aci\_data\_t \* received\_data);

static uint8\_t spi\_readwrite(uint8\_t aci\_byte);

static bool aci\_debug\_print = false;

aci\_queue\_t aci\_tx\_q;

aci\_queue\_t aci\_rx\_q;

static aci\_pins\_t \*a\_pins\_local\_ptr;

void m\_aci\_data\_print(hal\_aci\_data\_t \*p\_data)

{

 const uint8\_t length = p\_data->buffer[0];

 uint8\_t i;

 //Serial.print(length, DEC);

 printf(" %d ", length);

 //Serial.print(" :");

 printf(" :");

 for (i=0; i<=length; i++)

 {

 //Serial.print(p\_data->buffer[i], HEX);

// printf(" %p ", (void \*)p\_data->buffer[i]);

 //Serial.print(F(", "));

 printf(", ");

 }

 //Serial.println(F(""));

 printf("");

}

/\*

 Interrupt service routine called when the RDYN line goes low. Runs the SPI transfer.

\*/

/\*

 Checks the RDYN line and runs the SPI transfer if required.

\*/

static void m\_aci\_event\_check(void)

{

 hal\_aci\_data\_t data\_to\_send;

 hal\_aci\_data\_t received\_data;

 // No room to store incoming messages

 if (aci\_queue\_is\_full(&aci\_rx\_q))

 {

 return;

 }

 // If the ready line is disabled and we have pending messages outgoing we enable the request line

 if (PORTDbits.RD0 == 1)

 {

 if (!aci\_queue\_is\_empty(&aci\_tx\_q))

 {

 m\_aci\_reqn\_enable();

 }

 return;

 }

 // Receive from queue

 if (!aci\_queue\_dequeue(&aci\_tx\_q, &data\_to\_send))

 {

 /\* queue was empty, nothing to send \*/

 data\_to\_send.status\_byte = 0;

 data\_to\_send.buffer[0] = 0;

 }

 // Receive and/or transmit data

 m\_aci\_spi\_transfer(&data\_to\_send, &received\_data);

 /\* If there are messages to transmit, and we can store the reply, we request a new transfer \*/

 if (!aci\_queue\_is\_full(&aci\_rx\_q) && !aci\_queue\_is\_empty(&aci\_tx\_q))

 {

 m\_aci\_reqn\_enable();

 }

 // Check if we received data

 if (received\_data.buffer[0] > 0)

 {

 if (!aci\_queue\_enqueue(&aci\_rx\_q, &received\_data))

 {

 /\* Receive Buffer full.

 Should never happen.

 Spin in a while loop.

 \*/

 while(1);

 }

 }

 return;

}

/\*\* @brief Point the low level library at the ACI pins specified

 \* @details

 \* The ACI pins are specified in the application and a pointer is made available for

 \* the low level library to use

 \*/

static void m\_aci\_pins\_set(aci\_pins\_t \*a\_pins\_ptr)

{

 a\_pins\_local\_ptr = a\_pins\_ptr;

}

static inline void m\_aci\_reqn\_disable (void)

{

 //digitalWrite(a\_pins\_local\_ptr->reqn\_pin, 1);

 LATBbits.LATB8=1;

}

static inline void m\_aci\_reqn\_enable (void)

{

 //digitalWrite = (a\_pins\_local\_ptr->reqn\_pin, 0);

 LATBbits.LATB8=0;

}

static void m\_aci\_q\_flush(void)

{

 /\* re-initialize aci cmd queue and aci event queue to flush them\*/

 aci\_queue\_init(&aci\_tx\_q);

 aci\_queue\_init(&aci\_rx\_q);

}

static bool m\_aci\_spi\_transfer(hal\_aci\_data\_t \* data\_to\_send, hal\_aci\_data\_t \* received\_data)

{

 uint8\_t byte\_cnt;

 uint8\_t byte\_sent\_cnt;

 uint8\_t max\_bytes;

 m\_aci\_reqn\_enable();

 // Send length, receive header

 byte\_sent\_cnt = 0;

 //uint8\_t test = spi\_readwrite(data\_to\_send->buffer[byte\_sent\_cnt++]);

 //printf(" 0x%02x\n", test);

 received\_data->status\_byte = spi\_readwrite(data\_to\_send->buffer[byte\_sent\_cnt++]);

 //printf(" %p ",received\_data->status\_byte ); //FF 00 04

 // Send first byte, receive length from slave

 received\_data->buffer[0] = spi\_readwrite(data\_to\_send->buffer[byte\_sent\_cnt++]);

 //printf(" %p ",received\_data->buffer[0] );// 1 1 1 1 1 1 1 1 1 1 1 1 1 1

 if (0 == data\_to\_send->buffer[0])

 {

 max\_bytes = received\_data->buffer[0];

 }

 else

 {

 // Set the maximum to the biggest size. One command byte is already sent

 max\_bytes = (received\_data->buffer[0] > (data\_to\_send->buffer[0] - 1))

 ? received\_data->buffer[0]

 : (data\_to\_send->buffer[0] - 1);

 //if (received\_data->buffer[0] > (data\_to\_send->buffer[0] - 1))

 //{

 //max\_bytes = received\_data->buffer[0];

 //}

 // else

 //{

 //max\_bytes = (data\_to\_send->buffer[0] - 1);

 //}

}

 if (max\_bytes > HAL\_ACI\_MAX\_LENGTH)

 {

 max\_bytes = HAL\_ACI\_MAX\_LENGTH;

 }

 // Transmit/receive the rest of the packet

 for (byte\_cnt = 0; byte\_cnt < max\_bytes; byte\_cnt++)

 {

 received\_data->buffer[byte\_cnt+1] = spi\_readwrite(data\_to\_send->buffer[byte\_sent\_cnt++]); // changed to 0 instead of 1

 //printf(" %p Buffer Data, third transfer \n ",received\_data->buffer[byte\_cnt+1]);

 }

 // RDYN should follow the REQN line in approx 100ns

 m\_aci\_reqn\_disable();

 return (max\_bytes > 0);

}

void hal\_aci\_tl\_debug\_print(bool enable)

{

 aci\_debug\_print = enable;

}

void hal\_aci\_tl\_pin\_reset(void)

{

 if (UNUSED != a\_pins\_local\_ptr->reset\_pin)

 {

 //pinMode(a\_pins\_local\_ptr->reset\_pin, OUTPUT);

 /\*

 if ((REDBEARLAB\_SHIELD\_V1\_1 == a\_pins\_local\_ptr->board\_name) ||

 (REDBEARLAB\_SHIELD\_V2012\_07 == a\_pins\_local\_ptr->board\_name))

 {

 //The reset for the Redbearlab v1.1 and v2012.07 boards are inverted and has a Power On Reset

 //circuit that takes about 100ms to trigger the reset

 //digitalWrite(a\_pins\_local\_ptr->reset\_pin, 1);

 LATBbits.LATB12=1;

 delay\_ms(100);

 //digitalWrite(a\_pins\_local\_ptr->reset\_pin, 0);

 LATBbits.LATB12=0;

 //a\_pins\_local\_ptr->reset\_pin = 0;

 }\*/

 //else

 //{

 //digitalWrite(a\_pins\_local\_ptr->reset\_pin, 1);

 LATBbits.LATB12=1;

 //a\_pins\_local\_ptr->reset\_pin=1;

 //digitalWrite(a\_pins\_local\_ptr->reset\_pin, 0);

 LATBbits.LATB12 =0;

 //delay\_ms(1); // From Nordic Spec sheet (p48) reset needs to be held low minimum of 200ns. Too small to matter?

 //a\_pins\_local\_ptr->reset\_pin=0;

 //digitalWrite(a\_pins\_local\_ptr->reset\_pin, 1);

 LATBbits.LATB12=1;

 //a\_pins\_local\_ptr->reset\_pin=1;

 //}

 }

}

bool hal\_aci\_tl\_event\_peek(hal\_aci\_data\_t \*p\_aci\_data)

{

 // if (!a\_pins\_local\_ptr->interface\_is\_interrupt)

 //{

 m\_aci\_event\_check();

 //}

 if (aci\_queue\_peek(&aci\_rx\_q, p\_aci\_data))

 {

 return true;

 }

 return false;

}

bool hal\_aci\_tl\_event\_get(hal\_aci\_data\_t \*p\_aci\_data)

{

 bool was\_full;

 if (!aci\_queue\_is\_full(&aci\_rx\_q))

 {

 m\_aci\_event\_check();

 }

 was\_full = aci\_queue\_is\_full(&aci\_rx\_q);

 if (aci\_queue\_dequeue(&aci\_rx\_q, p\_aci\_data))

 {

 if (aci\_debug\_print)

 {

 //Serial.print(" E");

 printf("E");

 m\_aci\_data\_print(p\_aci\_data);

 }

 /\* Attempt to pull REQN LOW since we've made room for new messages \*/

 if (!aci\_queue\_is\_full(&aci\_rx\_q) && !aci\_queue\_is\_empty(&aci\_tx\_q))

 {

 m\_aci\_reqn\_enable();

 }

 return true;

 }

 return false;

}

void hal\_aci\_tl\_init(aci\_pins\_t \*a\_pins, bool debug)

{

 aci\_debug\_print = debug;

 /\* Needs to be called as the first thing for proper intialization\*/

 m\_aci\_pins\_set(a\_pins);

 /\*

 The SPI lines used are mapped directly to the hardware SPI

 MISO MOSI and SCK

 Change here if the pins are mapped differently

 The SPI library assumes that the hardware pins are used

 \*/

 SPIbegin();

 printf("SPI Begin\n");

 //Board dependent defines

 //#if defined (\_\_AVR\_\_)

 //For Arduino use the LSB first

 // SPI.setBitOrder(LSBFIRST);

 // #elif defined(\_\_PIC32MX\_\_)

 //For ChipKit use MSBFIRST and REVERSE the bits on the SPI as LSBFIRST is not supported

 //SPI.setBitOrder(MSBFIRST);

 // #endif

 //SPI.setClockDivider(a\_pins->spi\_clock\_divider); // taken care of in configbits

 //SPI.setDataMode(SPI\_MODE0); // taken care of with CKE, CKP, and SMP

 /\* Initialize the ACI Command queue. This must be called after the delay above. \*/

 aci\_queue\_init(&aci\_tx\_q);

 aci\_queue\_init(&aci\_rx\_q);

 //Configure the IO lines

 //pinMode(a\_pins->rdyn\_pin, INPUT\_PULLUP);

 //pinMode(a\_pins->reqn\_pin, OUTPUT);

 if (UNUSED != a\_pins->active\_pin)

 {

 TRISBbits.TRISB11=1;

 }

 /\* Pin reset the nRF8001, required when the nRF8001 setup is being changed \*/

 hal\_aci\_tl\_pin\_reset();

 printf("reset\n");

 /\* Set the nRF8001 to a known state as required by the datasheet\*/

 //digitalWrite(a\_pins->miso\_pin, 0);

 //SPI4BUF=0;

 //digitalWrite(a\_pins->mosi\_pin, 0);

 //SPI4BUF=0; // got rid of- what was causing the errors

 //a\_pins->miso\_pin = 0;

 //digitalWrite(a\_pins->reqn\_pin, 1);

 LATBbits.LATB8=1;

 //digitalWrite(a\_pins->sck\_pin, 0);

 //LATBbits.B14=0;

 //a\_pins->sck\_pin = 0;

 delay\_ms(60); //Wait for the nRF8001 to get hold of its lines - the lines float for a few ms after the reset

 /\*left at 60ms 12/18/14 Tried with 30, worked but 60 looked better on the logic analyzer\*/

}

bool hal\_aci\_tl\_send(hal\_aci\_data\_t \*p\_aci\_cmd)

{

 const uint8\_t length = p\_aci\_cmd->buffer[0];

 bool ret\_val = false;

 if (length > HAL\_ACI\_MAX\_LENGTH)

 {

 return false;

 }

 ret\_val = aci\_queue\_enqueue(&aci\_tx\_q, p\_aci\_cmd);

 if (ret\_val)

 {

 if(!aci\_queue\_is\_full(&aci\_rx\_q))

 {

 // Lower the REQN only when successfully enqueued

 m\_aci\_reqn\_enable();

 }

 if (aci\_debug\_print)

 {

 //Serial.print("C"); //ACI Command

 printf("C");

 m\_aci\_data\_print(p\_aci\_cmd);

 }

 }

 return ret\_val;

}

static uint8\_t spi\_readwrite(const uint8\_t aci\_byte)

{

 //Board dependent defines

//#if defined (\_\_AVR\_\_)

 //For Arduino the transmission does not have to be reversed

 // return SPI.transfer(aci\_byte);

//#elif defined(\_\_PIC32MX\_\_)

 //For ChipKit the transmission has to be reversed

 uint8\_t tmp\_bits;

 tmp\_bits = SPItransfer(REVERSE\_BITS(aci\_byte));

 return REVERSE\_BITS(tmp\_bits);

//#endif

}

bool hal\_aci\_tl\_rx\_q\_empty (void)

{

 return aci\_queue\_is\_empty(&aci\_rx\_q);

}

bool hal\_aci\_tl\_rx\_q\_full (void)

{

 return aci\_queue\_is\_full(&aci\_rx\_q);

}

bool hal\_aci\_tl\_tx\_q\_empty (void)

{

 return aci\_queue\_is\_empty(&aci\_tx\_q);

}

bool hal\_aci\_tl\_tx\_q\_full (void)

{

 return aci\_queue\_is\_full(&aci\_tx\_q);

}

void hal\_aci\_tl\_q\_flush (void)

{

 m\_aci\_q\_flush();

}

void SPIbegin(void)

{

 unsigned int rData;

 IEC0CLR=0x03800000; //disable all interrupts

 SPI4CON = 0; // Stops and resets the SPI1.

 rData=SPI4BUF; // clears the receive buffer

 SPI4BRG=4; // needs to be 4 so SPI clock will run at 2MHz (fPB/(2(SPI4BRG+1))=2)

 SPI4CONbits.CKE=1;// changed from 1 to 0?

 SPI4CONbits.CKP=0;

 SPI4CONbits.SMP=0; // changed from 1 to 0

 SPI4CONbits.MSTEN=1; // make sure the SPI is in master mode

 //SPI4BRG=0;

 SPI4STATbits.SPIROV=0; // clear the overflow bit

 SPI4CONbits.ON=1; // need to turn it on before initializing the EEPROM otherwise things get reset

 /\*Initializes the EEPROM\*/

 DDPCONbits.JTAGEN=0; // turns off the JTAG

 AD1PCFGbits.PCFG8=1; // Makes these bits be digital instead of analog outputs

 AD1PCFGbits.PCFG10=1;

 AD1PCFGbits.PCFG11=1;

 AD1PCFGbits.PCFG14=1;

 TRISBbits.TRISB8=0; // need B8 (the !CS line) to be an output

 TRISDbits.TRISD0=1; // make the ready pin an input

 LATBbits.LATB8=1; // pull the REQN (chip select) line high in the beginning

 TRISBbits.TRISB12=0; // need B12, the reset line, to be an output

 TRISBbits.TRISB11=1; // need B11, the active pin, to be an input

}

uint8\_t SPItransfer(uint8\_t reverse\_aci\_byte)

{

 IFS1bits.SPI4RXIF=0; // clear the interrupt flag (is this the RDYN pin)

 SPI4BUF=reverse\_aci\_byte; // loads the transmit buffer with new data

 while(!IFS1bits.SPI4RXIF); // waits for all the data to get shifted out by polling the interrupt flag

 uint8\_t newData=SPI4BUF; // get the new data and clear the buffer

 return newData; // return the new data

}

6. lib\_aci.c

#include "hal\_platform.h"

#include "aci.h"

#include "aci\_cmds.h"

#include "aci\_evts.h"

#include "aci\_protocol\_defines.h"

#include "acilib\_defs.h"

#include "acilib\_if.h"

#include "hal\_aci\_tl.h"

#include "aci\_queue.h"

#include "lib\_aci.h"

#include <stdio.h>

#include <stdbool.h>

#define LIB\_ACI\_DEFAULT\_CREDIT\_NUMBER 1

/\*

Global additionally used used in aci\_setup

\*/

hal\_aci\_data\_t msg\_to\_send;

static services\_pipe\_type\_mapping\_t \* p\_services\_pipe\_type\_map;

static hal\_aci\_data\_t \* p\_setup\_msgs;

static bool is\_request\_operation\_pending;

static bool is\_indicate\_operation\_pending;

static bool is\_open\_remote\_pipe\_pending;

static bool is\_close\_remote\_pipe\_pending;

static uint8\_t request\_operation\_pipe = 0;

static uint8\_t indicate\_operation\_pipe = 0;

// The following structure (aci\_cmd\_params\_open\_adv\_pipe) will be used to store the complete command

// including the pipes to be opened.

static aci\_cmd\_params\_open\_adv\_pipe\_t aci\_cmd\_params\_open\_adv\_pipe;

extern aci\_queue\_t aci\_rx\_q;

extern aci\_queue\_t aci\_tx\_q;

bool lib\_aci\_is\_pipe\_available(aci\_state\_t \*aci\_stat, uint8\_t pipe)

{

 uint8\_t byte\_idx;

 byte\_idx = pipe / 8;

 if (aci\_stat->pipes\_open\_bitmap[byte\_idx] & (0x01 << (pipe % 8)))

 {

 return(true);

 }

 return(false);

}

bool lib\_aci\_is\_pipe\_closed(aci\_state\_t \*aci\_stat, uint8\_t pipe)

{

 uint8\_t byte\_idx;

 byte\_idx = pipe / 8;

 if (aci\_stat->pipes\_closed\_bitmap[byte\_idx] & (0x01 << (pipe % 8)))

 {

 return(true);

 }

 return(false);

}

bool lib\_aci\_is\_discovery\_finished(aci\_state\_t \*aci\_stat)

{

 return(aci\_stat->pipes\_open\_bitmap[0]&0x01);

}

void lib\_aci\_board\_init(aci\_state\_t \*aci\_stat)

{

 hal\_aci\_evt\_t \*aci\_data = NULL;

 aci\_data = (hal\_aci\_evt\_t \*)&msg\_to\_send;

 //if (REDBEARLAB\_SHIELD\_V1\_1 == aci\_stat->aci\_pins.board\_name)

 //{

 /\*

 The Bluetooth low energy Arduino shield v1.1 requires about 100ms to reset.

 This is not required for the nRF2740, nRF2741 modules

 \*/

 //delay(100);

 /\*

 Send the soft reset command to the nRF8001 to get the nRF8001 to a known state.

 \*/

 //lib\_aci\_radio\_reset();

 //while (1)

 //{

 /\*Wait for the command response of the radio reset command.

 as the nRF8001 will be in either SETUP or STANDBY after the ACI Reset Radio is processed

 \*/

 //if (true == lib\_aci\_event\_get(aci\_stat, aci\_data))

 //{

 // aci\_evt\_t \* aci\_evt;

 //aci\_evt = &(aci\_data->evt);

 //if (ACI\_EVT\_CMD\_RSP == aci\_evt->evt\_opcode)

 //{

 //if (ACI\_STATUS\_ERROR\_DEVICE\_STATE\_INVALID == aci\_evt->params.cmd\_rsp.cmd\_status) //in SETUP

 //{

 //Inject a Device Started Event Setup to the ACI Event Queue

 //msg\_to\_send.buffer[0] = 4; //Length

 //msg\_to\_send.buffer[1] = 0x81; //Device Started Event

 //msg\_to\_send.buffer[2] = 0x02; //Setup

 //msg\_to\_send.buffer[3] = 0; //Hardware Error -> None

 //msg\_to\_send.buffer[4] = 2; //Data Credit Available

 //aci\_queue\_enqueue(&aci\_rx\_q, &msg\_to\_send);

 //}

 //else if (ACI\_STATUS\_SUCCESS == aci\_evt->params.cmd\_rsp.cmd\_status) //We are now in STANDBY

 //{

 //Inject a Device Started Event Standby to the ACI Event Queue

 //msg\_to\_send.buffer[0] = 4; //Length

 //msg\_to\_send.buffer[1] = 0x81; //Device Started Event

 //msg\_to\_send.buffer[2] = 0x03; //Standby

 //msg\_to\_send.buffer[3] = 0; //Hardware Error -> None

 //msg\_to\_send.buffer[4] = 2; //Data Credit Available

 //aci\_queue\_enqueue(&aci\_rx\_q, &msg\_to\_send);

 //}

 //else if (ACI\_STATUS\_ERROR\_CMD\_UNKNOWN == aci\_evt->params.cmd\_rsp.cmd\_status) //We are now in TEST

 //{

 //Inject a Device Started Event Test to the ACI Event Queue

 //msg\_to\_send.buffer[0] = 4; //Length

 //msg\_to\_send.buffer[1] = 0x81; //Device Started Event

 //msg\_to\_send.buffer[2] = 0x01; //Test

 //msg\_to\_send.buffer[3] = 0; //Hardware Error -> None

 //msg\_to\_send.buffer[4] = 0; //Data Credit Available

 //aci\_queue\_enqueue(&aci\_rx\_q, &msg\_to\_send);

 //}

 //Break out of the while loop

 //break;

 //}

 //else

 //{

 //Serial.println(F("Discard any other ACI Events"));

 // }

 //}

 // }

 //}

}

void lib\_aci\_init(aci\_state\_t \*aci\_stat, bool debug)

{

 uint8\_t i;

 for (i = 0; i < PIPES\_ARRAY\_SIZE; i++)

 {

 aci\_stat->pipes\_open\_bitmap[i] = 0;

 aci\_stat->pipes\_closed\_bitmap[i] = 0;

 aci\_cmd\_params\_open\_adv\_pipe.pipes[i] = 0;

 }

 is\_request\_operation\_pending = false;

 is\_indicate\_operation\_pending = false;

 is\_open\_remote\_pipe\_pending = false;

 is\_close\_remote\_pipe\_pending = false;

 request\_operation\_pipe = 0;

 indicate\_operation\_pipe = 0;

 p\_services\_pipe\_type\_map = aci\_stat->aci\_setup\_info.services\_pipe\_type\_mapping;

 p\_setup\_msgs = aci\_stat->aci\_setup\_info.setup\_msgs;

 hal\_aci\_tl\_init(&aci\_stat->aci\_pins, debug);

 lib\_aci\_board\_init(aci\_stat);

}

uint8\_t lib\_aci\_get\_nb\_available\_credits(aci\_state\_t \*aci\_stat)

{

 return aci\_stat->data\_credit\_available;

}

uint16\_t lib\_aci\_get\_cx\_interval\_ms(aci\_state\_t \*aci\_stat)

{

 uint32\_t cx\_rf\_interval\_ms\_32bits;

 uint16\_t cx\_rf\_interval\_ms;

 cx\_rf\_interval\_ms\_32bits = aci\_stat->connection\_interval;

 cx\_rf\_interval\_ms\_32bits \*= 125; // the connection interval is given in multiples of 0.125 milliseconds

 cx\_rf\_interval\_ms = cx\_rf\_interval\_ms\_32bits / 100;

 return cx\_rf\_interval\_ms;

}

uint16\_t lib\_aci\_get\_cx\_interval(aci\_state\_t \*aci\_stat)

{

 return aci\_stat->connection\_interval;

}

uint16\_t lib\_aci\_get\_slave\_latency(aci\_state\_t \*aci\_stat)

{

 return aci\_stat->slave\_latency;

}

bool lib\_aci\_set\_app\_latency(uint16\_t latency, aci\_app\_latency\_mode\_t latency\_mode)

{

 aci\_cmd\_params\_set\_app\_latency\_t aci\_set\_app\_latency;

 aci\_set\_app\_latency.mode = latency\_mode;

 aci\_set\_app\_latency.latency = latency;

 acil\_encode\_cmd\_set\_app\_latency(&(msg\_to\_send.buffer[0]), &aci\_set\_app\_latency);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_test(aci\_test\_mode\_change\_t enter\_exit\_test\_mode)

{

 aci\_cmd\_params\_test\_t aci\_cmd\_params\_test;

 aci\_cmd\_params\_test.test\_mode\_change = enter\_exit\_test\_mode;

 acil\_encode\_cmd\_set\_test\_mode(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_test);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_sleep()

{

 acil\_encode\_cmd\_sleep(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_radio\_reset()

{

 acil\_encode\_baseband\_reset(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_direct\_connect()

{

 acil\_encode\_direct\_connect(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_device\_version()

{

 acil\_encode\_cmd\_get\_device\_version(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_set\_local\_data(aci\_state\_t \*aci\_stat, uint8\_t pipe, uint8\_t \*p\_value, uint8\_t size)

{

 aci\_cmd\_params\_set\_local\_data\_t aci\_cmd\_params\_set\_local\_data;

 if ((p\_services\_pipe\_type\_map[pipe-1].location != ACI\_STORE\_LOCAL)

 ||

 (size > ACI\_PIPE\_TX\_DATA\_MAX\_LEN))

 {

 return false;

 }

 aci\_cmd\_params\_set\_local\_data.tx\_data.pipe\_number = pipe;

 memcpy(&(aci\_cmd\_params\_set\_local\_data.tx\_data.aci\_data[0]), p\_value, size);

 acil\_encode\_cmd\_set\_local\_data(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_set\_local\_data, size);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_connect(uint16\_t run\_timeout, uint16\_t adv\_interval)

{

 aci\_cmd\_params\_connect\_t aci\_cmd\_params\_connect;

 aci\_cmd\_params\_connect.timeout = run\_timeout;

 aci\_cmd\_params\_connect.adv\_interval = adv\_interval;

 acil\_encode\_cmd\_connect(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_connect);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_disconnect(aci\_state\_t \*aci\_stat, aci\_disconnect\_reason\_t reason)

{

 bool ret\_val;

 uint8\_t i;

 aci\_cmd\_params\_disconnect\_t aci\_cmd\_params\_disconnect;

 aci\_cmd\_params\_disconnect.reason = reason;

 acil\_encode\_cmd\_disconnect(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_disconnect);

 ret\_val = hal\_aci\_tl\_send(&msg\_to\_send);

 // If we have actually sent the disconnect

 if (ret\_val)

 {

 // Update pipes immediately so that while the disconnect is happening,

 // the application can't attempt sending another message

 // If the application sends another message before we updated this

 // a ACI Pipe Error Event will be received from nRF8001

 for (i=0; i < PIPES\_ARRAY\_SIZE; i++)

 {

 aci\_stat->pipes\_open\_bitmap[i] = 0;

 aci\_stat->pipes\_closed\_bitmap[i] = 0;

 }

 }

 return ret\_val;

}

bool lib\_aci\_bond(uint16\_t run\_timeout, uint16\_t adv\_interval)

{

 aci\_cmd\_params\_bond\_t aci\_cmd\_params\_bond;

 aci\_cmd\_params\_bond.timeout = run\_timeout;

 aci\_cmd\_params\_bond.adv\_interval = adv\_interval;

 acil\_encode\_cmd\_bond(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_bond);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_wakeup()

{

 acil\_encode\_cmd\_wakeup(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_set\_tx\_power(aci\_device\_output\_power\_t tx\_power)

{

 aci\_cmd\_params\_set\_tx\_power\_t aci\_cmd\_params\_set\_tx\_power;

 aci\_cmd\_params\_set\_tx\_power.device\_power = tx\_power;

 acil\_encode\_cmd\_set\_radio\_tx\_power(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_set\_tx\_power);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_get\_address()

{

 acil\_encode\_cmd\_get\_address(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_get\_temperature()

{

 acil\_encode\_cmd\_temparature(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_get\_battery\_level()

{

 acil\_encode\_cmd\_battery\_level(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_send\_data(uint8\_t pipe, uint8\_t \*p\_value, uint8\_t size)

{

 bool ret\_val = false;

 aci\_cmd\_params\_send\_data\_t aci\_cmd\_params\_send\_data;

 if(!((p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_TX) ||

 (p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_TX\_ACK)))

 {

 return false;

 }

 if (size > ACI\_PIPE\_TX\_DATA\_MAX\_LEN)

 {

 return false;

 }

 {

 aci\_cmd\_params\_send\_data.tx\_data.pipe\_number = pipe;

 memcpy(&(aci\_cmd\_params\_send\_data.tx\_data.aci\_data[0]), p\_value, size);

 acil\_encode\_cmd\_send\_data(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_send\_data, size);

 ret\_val = hal\_aci\_tl\_send(&msg\_to\_send);

 }

 return ret\_val;

}

bool lib\_aci\_request\_data(aci\_state\_t \*aci\_stat, uint8\_t pipe)

{

 bool ret\_val = false;

 aci\_cmd\_params\_request\_data\_t aci\_cmd\_params\_request\_data;

 if(!((p\_services\_pipe\_type\_map[pipe-1].location == ACI\_STORE\_REMOTE)&&(p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_RX\_REQ)))

 {

 return false;

 }

 {

 {

 aci\_cmd\_params\_request\_data.pipe\_number = pipe;

 acil\_encode\_cmd\_request\_data(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_request\_data);

 ret\_val = hal\_aci\_tl\_send(&msg\_to\_send);

 }

 }

 return ret\_val;

}

bool lib\_aci\_change\_timing(uint16\_t minimun\_cx\_interval, uint16\_t maximum\_cx\_interval, uint16\_t slave\_latency, uint16\_t timeout)

{

 aci\_cmd\_params\_change\_timing\_t aci\_cmd\_params\_change\_timing;

 aci\_cmd\_params\_change\_timing.conn\_params.min\_conn\_interval = minimun\_cx\_interval;

 aci\_cmd\_params\_change\_timing.conn\_params.max\_conn\_interval = maximum\_cx\_interval;

 aci\_cmd\_params\_change\_timing.conn\_params.slave\_latency = slave\_latency;

 aci\_cmd\_params\_change\_timing.conn\_params.timeout\_mult = timeout;

 acil\_encode\_cmd\_change\_timing\_req(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_change\_timing);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_change\_timing\_GAP\_PPCP()

{

 acil\_encode\_cmd\_change\_timing\_req\_GAP\_PPCP(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_open\_remote\_pipe(aci\_state\_t \*aci\_stat, uint8\_t pipe)

{

 bool ret\_val = false;

 aci\_cmd\_params\_open\_remote\_pipe\_t aci\_cmd\_params\_open\_remote\_pipe;

 if(!((p\_services\_pipe\_type\_map[pipe-1].location == ACI\_STORE\_REMOTE)&&

 ((p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_RX)||

 (p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_RX\_ACK\_AUTO)||

 (p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_RX\_ACK))))

 {

 return false;

 }

 {

 is\_request\_operation\_pending = true;

 is\_open\_remote\_pipe\_pending = true;

 request\_operation\_pipe = pipe;

 aci\_cmd\_params\_open\_remote\_pipe.pipe\_number = pipe;

 acil\_encode\_cmd\_open\_remote\_pipe(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_open\_remote\_pipe);

 ret\_val = hal\_aci\_tl\_send(&msg\_to\_send);

 }

 return ret\_val;

}

bool lib\_aci\_close\_remote\_pipe(aci\_state\_t \*aci\_stat, uint8\_t pipe)

{

 bool ret\_val = false;

 aci\_cmd\_params\_close\_remote\_pipe\_t aci\_cmd\_params\_close\_remote\_pipe;

 if(!((p\_services\_pipe\_type\_map[pipe-1].location == ACI\_STORE\_REMOTE)&&

 ((p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_RX)||

 (p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_RX\_ACK\_AUTO)||

 (p\_services\_pipe\_type\_map[pipe-1].pipe\_type == ACI\_RX\_ACK))))

 {

 return false;

 }

 {

 is\_request\_operation\_pending = true;

 is\_close\_remote\_pipe\_pending = true;

 request\_operation\_pipe = pipe;

 aci\_cmd\_params\_close\_remote\_pipe.pipe\_number = pipe;

 acil\_encode\_cmd\_close\_remote\_pipe(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_close\_remote\_pipe);

 ret\_val = hal\_aci\_tl\_send(&msg\_to\_send);

 }

 return ret\_val;

}

bool lib\_aci\_set\_key(aci\_key\_type\_t key\_rsp\_type, uint8\_t \*key, uint8\_t len)

{

 aci\_cmd\_params\_set\_key\_t aci\_cmd\_params\_set\_key;

 aci\_cmd\_params\_set\_key.key\_type = key\_rsp\_type;

 memcpy((uint8\_t\*)&(aci\_cmd\_params\_set\_key.key), key, len);

 acil\_encode\_cmd\_set\_key(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_set\_key);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_echo\_msg(uint8\_t msg\_size, uint8\_t \*p\_msg\_data)

{

 aci\_cmd\_params\_echo\_t aci\_cmd\_params\_echo;

 if(msg\_size > (ACI\_ECHO\_DATA\_MAX\_LEN))

 {

 return false;

 }

 if (msg\_size > (ACI\_ECHO\_DATA\_MAX\_LEN))

 {

 msg\_size = ACI\_ECHO\_DATA\_MAX\_LEN;

 }

 memcpy(&(aci\_cmd\_params\_echo.echo\_data[0]), p\_msg\_data, msg\_size);

 acil\_encode\_cmd\_echo\_msg(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_echo, msg\_size);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_bond\_request()

{

 acil\_encode\_cmd\_bond\_security\_request(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_event\_peek(hal\_aci\_evt\_t \*p\_aci\_evt\_data)

{

 return hal\_aci\_tl\_event\_peek((hal\_aci\_data\_t \*)p\_aci\_evt\_data);

}

bool lib\_aci\_event\_get(aci\_state\_t \*aci\_stat, hal\_aci\_evt\_t \*p\_aci\_evt\_data)

{

 bool status = false;

 status = hal\_aci\_tl\_event\_get((hal\_aci\_data\_t \*)p\_aci\_evt\_data);

 // printf ("0x%02x",status);

 /\*\*

 Update the state of the ACI with the

 ACI Events -> Pipe Status, Disconnected, Connected, Bond Status, Pipe Error

 \*/

 if (true == status)

 {

 aci\_evt\_t \* aci\_evt;

 aci\_evt = &p\_aci\_evt\_data->evt;

 switch(aci\_evt->evt\_opcode)

 {

 case ACI\_EVT\_PIPE\_STATUS:

 {

 uint8\_t i=0;

 for (i=0; i < PIPES\_ARRAY\_SIZE; i++)

 {

 aci\_stat->pipes\_open\_bitmap[i] = aci\_evt->params.pipe\_status.pipes\_open\_bitmap[i];

 aci\_stat->pipes\_closed\_bitmap[i] = aci\_evt->params.pipe\_status.pipes\_closed\_bitmap[i];

 }

 }

 break;

 case ACI\_EVT\_DISCONNECTED:

 {

 uint8\_t i=0;

 for (i=0; i < PIPES\_ARRAY\_SIZE; i++)

 {

 aci\_stat->pipes\_open\_bitmap[i] = 0;

 aci\_stat->pipes\_closed\_bitmap[i] = 0;

 }

 aci\_stat->confirmation\_pending = false;

 aci\_stat->data\_credit\_available = aci\_stat->data\_credit\_total;

 }

 break;

 case ACI\_EVT\_TIMING:

 aci\_stat->connection\_interval = aci\_evt->params.timing.conn\_rf\_interval;

 aci\_stat->slave\_latency = aci\_evt->params.timing.conn\_slave\_rf\_latency;

 aci\_stat->supervision\_timeout = aci\_evt->params.timing.conn\_rf\_timeout;

 break;

 default:

 /\* Need default case to avoid compiler warnings about missing enum

 \* values on some platforms.

 \*/

 break;

 }

 }

 return status;

}

bool lib\_aci\_send\_ack(aci\_state\_t \*aci\_stat, const uint8\_t pipe)

{

 bool ret\_val = false;

 {

 acil\_encode\_cmd\_send\_data\_ack(&(msg\_to\_send.buffer[0]), pipe);

 ret\_val = hal\_aci\_tl\_send(&msg\_to\_send);

 }

 return ret\_val;

}

bool lib\_aci\_send\_nack(aci\_state\_t \*aci\_stat, const uint8\_t pipe, const uint8\_t error\_code)

{

 bool ret\_val = false;

 {

 acil\_encode\_cmd\_send\_data\_nack(&(msg\_to\_send.buffer[0]), pipe, error\_code);

 ret\_val = hal\_aci\_tl\_send(&msg\_to\_send);

 }

 return ret\_val;

}

bool lib\_aci\_broadcast(const uint16\_t timeout, const uint16\_t adv\_interval)

{

 aci\_cmd\_params\_broadcast\_t aci\_cmd\_params\_broadcast;

 if (timeout > 16383)

 {

 return false;

 }

 // The adv\_interval should be between 160 and 16384 (which translates to the advertisement

 // interval values 100 ms and 10.24 s.

 if ((160 > adv\_interval) || (adv\_interval > 16384))

 {

 return false;

 }

 aci\_cmd\_params\_broadcast.timeout = timeout;

 aci\_cmd\_params\_broadcast.adv\_interval = adv\_interval;

 acil\_encode\_cmd\_broadcast(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_broadcast);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_open\_adv\_pipes(const uint8\_t \* const adv\_service\_data\_pipes)

{

 uint8\_t i;

 for (i = 0; i < PIPES\_ARRAY\_SIZE; i++)

 {

 aci\_cmd\_params\_open\_adv\_pipe.pipes[i] = adv\_service\_data\_pipes[i];

 }

 acil\_encode\_cmd\_open\_adv\_pipes(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_open\_adv\_pipe);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_open\_adv\_pipe(const uint8\_t pipe)

{

 uint8\_t byte\_idx = pipe / 8;

 aci\_cmd\_params\_open\_adv\_pipe.pipes[byte\_idx] |= (0x01 << (pipe % 8));

 acil\_encode\_cmd\_open\_adv\_pipes(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_open\_adv\_pipe);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_read\_dynamic\_data()

{

 acil\_encode\_cmd\_read\_dynamic\_data(&(msg\_to\_send.buffer[0]));

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_write\_dynamic\_data(uint8\_t sequence\_number, uint8\_t\* dynamic\_data, uint8\_t length)

{

 acil\_encode\_cmd\_write\_dynamic\_data(&(msg\_to\_send.buffer[0]), sequence\_number, dynamic\_data, length);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

bool lib\_aci\_dtm\_command(uint8\_t dtm\_command\_msbyte, uint8\_t dtm\_command\_lsbyte)

{

 aci\_cmd\_params\_dtm\_cmd\_t aci\_cmd\_params\_dtm\_cmd;

 aci\_cmd\_params\_dtm\_cmd.cmd\_msb = dtm\_command\_msbyte;

 aci\_cmd\_params\_dtm\_cmd.cmd\_lsb = dtm\_command\_lsbyte;

 acil\_encode\_cmd\_dtm\_cmd(&(msg\_to\_send.buffer[0]), &aci\_cmd\_params\_dtm\_cmd);

 return hal\_aci\_tl\_send(&msg\_to\_send);

}

void lib\_aci\_flush(void)

{

 hal\_aci\_tl\_q\_flush();

}

void lib\_aci\_debug\_print(bool enable)

{

 hal\_aci\_tl\_debug\_print(enable);

}

void lib\_aci\_pin\_reset(void)

{

 hal\_aci\_tl\_pin\_reset();

}

bool lib\_aci\_event\_queue\_empty(void)

{

 return hal\_aci\_tl\_rx\_q\_empty();

}

bool lib\_aci\_event\_queue\_full(void)

{

 return hal\_aci\_tl\_rx\_q\_full();

}

bool lib\_aci\_command\_queue\_empty(void)

{

 return hal\_aci\_tl\_tx\_q\_empty();

}

bool lib\_aci\_command\_queue\_full(void)

{

 return hal\_aci\_tl\_tx\_q\_full();

}

ii. header files:

1. aci.h

#ifndef ACI\_H\_\_

#define ACI\_H\_\_

#include <stdint.h>

#include <stdbool.h>

//#define true 1

//#define false 0

//#define bool \_Bool // need this since converting from C++ code

/\*\*

 \* Define an \_aci\_packed\_ macro we can use in structure and enumerated type

 \* declarations so that the types are sized consistently across different

 \* platforms. In particular Arduino platforms using the GCC compiler and the

 \* Nordic processors using the Keil compiler.

 \*

 \* It's really the GNU compiler platforms that need a special keyword to get

 \* tight packing of values. On GNU platforms we can use the keyword:

 \* \_\_attribute\_\_((\_\_packed\_\_))

 \* The thing is that while this keyword does the right thing with old and new

 \* versions of the gcc (C) compiler it only works right with g++ (C++) compiler

 \* versions that are version 4 or newer.

 \*/

#ifdef \_\_GNUC\_\_

# if \_\_GNUC\_\_ >= 4

# define \_aci\_packed\_ \_\_attribute\_\_((\_\_packed\_\_))

# else

# error "older g++ versions don't handle packed attribute in typedefs"

# endif

#else

# define \_aci\_packed\_

#endif

/\*

 \* Define a macro that compares the size of the first parameter to the integer

 \* value of the second parameter. If they do not match, a compile time error

 \* for negative array size occurs (even gnu chokes on negative array size).

 \*

 \* This compare is done by creating a typedef for an array. No variables are

 \* created and no memory is consumed with this check. The created type is

 \* used for checking only and is not for use by any other code. The value

 \* of 10 in this macro is arbitrary, it just needs to be a value larger

 \* than one to result in a positive number for the array size.

 \*/

#define ACI\_ASSERT\_SIZE(x,y) typedef char x ## \_assert\_size\_t[-1+10\*(sizeof(x) == (y))]

/\*\*

 \* @def ACI\_VERSION

 \* @brief Current ACI protocol version. 0 means a device that is not yet released.

 \* A numer greater than 0 refers to a specific ACI version documented and released.

 \* The ACI consists of the ACI commands, ACI events and error codes.

 \*/

#define ACI\_VERSION (0x02)

/\*\*

 \* @def BTLE\_DEVICE\_ADDRESS\_SIZE

 \* @brief Size in bytes of a Bluetooth Address

 \*/

#define BTLE\_DEVICE\_ADDRESS\_SIZE (6)

/\*\*

 \* @def ACI\_PACKET\_MAX\_LEN

 \* @brief Maximum length in bytes of a full ACI packet, including length prefix, opcode and payload

 \*/

#define ACI\_PACKET\_MAX\_LEN (32)

/\*\*

 \* @def ACI\_ECHO\_DATA\_MAX\_LEN

 \* @brief Maximum length in bytes of the echo data portion

 \*/

#define ACI\_ECHO\_DATA\_MAX\_LEN (ACI\_PACKET\_MAX\_LEN - 3)

/\*\*

 \* @def ACI\_DEVICE\_MAX\_PIPES

 \* @brief Maximum number of ACI pipes

 \*/

#define ACI\_DEVICE\_MAX\_PIPES (62)

/\*\*

 \* @def ACI\_PIPE\_TX\_DATA\_MAX\_LEN

 \* @brief Maximum length in bytes of a transmission data pipe packet

 \*/

#define ACI\_PIPE\_TX\_DATA\_MAX\_LEN (20)

/\*\*

 \* @def ACI\_PIPE\_RX\_DATA\_MAX\_LEN

 \* @brief Maximum length in bytes of a reception data pipe packet

 \*/

#define ACI\_PIPE\_RX\_DATA\_MAX\_LEN (22)

/\*\*

 \* @def ACI\_GAP\_DEVNAME\_MAX\_LEN

 \* @brief Maximum length in bytes of the GAP device name

 \*/

#define ACI\_GAP\_DEVNAME\_MAX\_LEN (20)

/\*\*

 \* @def ACI\_AD\_PACKET\_MAX\_LEN

 \* @brief Maximum length in bytes of an AD packet

 \*/

#define ACI\_AD\_PACKET\_MAX\_LEN (31)

/\*\*

 \* @def ACI\_AD\_PACKET\_MAX\_USER\_LEN

 \* @brief Maximum usable length in bytes of an AD packet

 \*/

#define ACI\_AD\_PACKET\_MAX\_USER\_LEN (31 - 3)

/\*\*

 \* @def ACI\_PIPE\_INVALID

 \* @brief Invalid pipe number

 \*/

#define ACI\_PIPE\_INVALID (0xFF)

/\*\*

 \* @enum aci\_pipe\_store\_t

 \* @brief Storage type identifiers: local and remote

 \*/

typedef enum

{

 ACI\_STORE\_INVALID = 0x0,

 ACI\_STORE\_LOCAL= 0x01,

 ACI\_STORE\_REMOTE= 0x02

} \_aci\_packed\_ aci\_pipe\_store\_t;

/\*\*

 \* @enum aci\_pipe\_type\_t

 \* @brief Pipe types

 \*/

typedef enum

{

 ACI\_TX\_BROADCAST = 0x0001,

 ACI\_TX = 0x0002,

 ACI\_TX\_ACK = 0x0004,

 ACI\_RX = 0x0008,

 ACI\_RX\_ACK = 0x0010,

 ACI\_TX\_REQ = 0x0020,

 ACI\_RX\_REQ = 0x0040,

 ACI\_SET = 0x0080,

 ACI\_TX\_SIGN = 0x0100,

 ACI\_RX\_SIGN = 0x0200,

 ACI\_RX\_ACK\_AUTO = 0x0400

} \_aci\_packed\_ aci\_pipe\_type\_t;

ACI\_ASSERT\_SIZE(aci\_pipe\_type\_t, 2);

/\*\*

 \* @enum aci\_bd\_addr\_type\_t

 \* @brief Bluetooth Address types

 \*/

typedef enum

{

 ACI\_BD\_ADDR\_TYPE\_INVALID = 0x00,

 ACI\_BD\_ADDR\_TYPE\_PUBLIC = 0x01,

 ACI\_BD\_ADDR\_TYPE\_RANDOM\_STATIC = 0x02,

 ACI\_BD\_ADDR\_TYPE\_RANDOM\_PRIVATE\_RESOLVABLE = 0x03,

 ACI\_BD\_ADDR\_TYPE\_RANDOM\_PRIVATE\_UNRESOLVABLE = 0x04

} \_aci\_packed\_ aci\_bd\_addr\_type\_t;

/\*\*

 \* @enum aci\_device\_output\_power\_t

 \* @brief Radio output power levels

 \*/

typedef enum

{

 ACI\_DEVICE\_OUTPUT\_POWER\_MINUS\_18DBM = 0x00, /\*\*< Output power set to -18dBm \*/

 ACI\_DEVICE\_OUTPUT\_POWER\_MINUS\_12DBM = 0x01, /\*\*< Output power set to -12dBm \*/

 ACI\_DEVICE\_OUTPUT\_POWER\_MINUS\_6DBM = 0x02, /\*\*< Output power set to -6dBm \*/

 ACI\_DEVICE\_OUTPUT\_POWER\_0DBM = 0x03 /\*\*< Output power set to 0dBm - DEFAULT\*/

} \_aci\_packed\_ aci\_device\_output\_power\_t;

/\*\*

 \* @enum aci\_device\_operation\_mode\_t

 \* @brief Device operation modes

 \*/

typedef enum

{

 ACI\_DEVICE\_INVALID =0x00,

 ACI\_DEVICE\_TEST =0x01,

 ACI\_DEVICE\_SETUP =0x02,

 ACI\_DEVICE\_STANDBY =0x03,

 ACI\_DEVICE\_SLEEP =0x04

} \_aci\_packed\_ aci\_device\_operation\_mode\_t;

/\*\*

 \* @enum aci\_disconnect\_reason\_t

 \* @brief Reason enumeration for ACI\_CMD\_DISCONNECT

 \*/

typedef enum

{

 ACI\_REASON\_TERMINATE =0x01, /\*\*< Use this to disconnect (does a terminate request), you need to wait for the "disconnected" event \*/

 ACI\_REASON\_BAD\_TIMING =0x02 /\*<Use this to disconnect and inform the peer, that the timing on the link is not acceptable for the device, you need to wait for the "disconnected" event \*/

} \_aci\_packed\_ aci\_disconnect\_reason\_t;

/\*\*

 \* @enum aci\_test\_mode\_change\_t

 \* @brief Device test mode control

 \*/

typedef enum

{

 ACI\_TEST\_MODE\_DTM\_UART = 0x01,

 ACI\_TEST\_MODE\_DTM\_ACI = 0x02,

 ACI\_TEST\_MODE\_EXIT = 0xFF

} \_aci\_packed\_ aci\_test\_mode\_change\_t;

ACI\_ASSERT\_SIZE(aci\_test\_mode\_change\_t, 1);

/\*\*

 \* @enum aci\_permissions\_t

 \* @brief Data store permissions

 \*/

typedef enum

{

 ACI\_PERMISSIONS\_NONE =0x00,

 ACI\_PERMISSIONS\_LINK\_AUTHENTICATED =0x01

} \_aci\_packed\_ aci\_permissions\_t;

/\*\*

 \* @def ACI\_VS\_UUID\_128\_MAX\_COUNT

 \* @brief Maximum number of 128-bit Vendor Specific

 \* UUIDs that can be set

 \*/

#define ACI\_VS\_UUID\_128\_MAX\_COUNT 64 /\*\* #0 reserved for invalid, #1 reservered for BT SIG and a maximum of 1024 bytes (16\*64) \*/

/\*\*

 \* @struct aci\_ll\_conn\_params\_t

 \* @brief Link Layer Connection Parameters

 \*/

typedef struct

{

 uint16\_t min\_conn\_interval; /\*\*< Minimum connection interval requested from peer \*/

 #define ACI\_PPCP\_MIN\_CONN\_INTVL\_NONE 0xFFFF

 #define ACI\_PPCP\_MIN\_CONN\_INTVL\_MIN 0x0006

 #define ACI\_PPCP\_MIN\_CONN\_INTVL\_MAX 0x0C80

 uint16\_t max\_conn\_interval; /\*\*< Maximum connection interval requested from peer \*/

 #define ACI\_PPCP\_MAX\_CONN\_INTVL\_NONE 0xFFFF

 #define ACI\_PPCP\_MAX\_CONN\_INTVL\_MIN 0x0006

 #define ACI\_PPCP\_MAX\_CONN\_INTVL\_MAX 0x0C80

 uint16\_t slave\_latency; /\*\*< Connection interval latency requested from peer \*/

 #define ACI\_PPCP\_SLAVE\_LATENCY\_MAX 0x03E8

 uint16\_t timeout\_mult; /\*\*< Link supervisor timeout multiplier requested from peer \*/

 #define ACI\_PPCP\_TIMEOUT\_MULT\_NONE 0xFFFF

 #define ACI\_PPCP\_TIMEOUT\_MULT\_MIN 0x000A

 #define ACI\_PPCP\_TIMEOUT\_MULT\_MAX 0x0C80

} \_aci\_packed\_ aci\_ll\_conn\_params\_t;

/\*\*

 \* @def aci\_gap\_ppcp\_t

 \* @brief GAP Peripheral Preferred Connection Parameters

 \*/

#define aci\_gap\_ppcp\_t aci\_ll\_conn\_params\_t

/\*\*

 \* @def ACI\_AD\_LOC\_SVCUUID\_16\_MAX\_COUNT

 \* @brief Maximum number of 16-bit UUIDs that can

 \* be inserted in the Services tag of AD

 \*/

#define ACI\_AD\_LOC\_SVCUUID\_16\_MAX\_COUNT 5

/\*\*

 \* @def ACI\_AD\_LOC\_SVCUUID\_128\_MAX\_COUNT

 \* @brief Maximum number of 128-bit UUIDs that can

 \* be inserted in the Services tag of AD

 \*/

#define ACI\_AD\_LOC\_SVCUUID\_128\_MAX\_COUNT 1

/\*\*

 \* @def ACI\_AD\_SOL\_SVCUUID\_16\_MAX\_COUNT

 \* @brief Maximum number of UUIDs that can

 \* be inserted in the Solicited Services tag of AD

 \*/

#define ACI\_AD\_SOL\_SVCUUID\_16\_MAX\_COUNT 5

/\*\*

 \* @def ACI\_AD\_SOL\_SVCUUID\_128\_MAX\_COUNT

 \* @brief Maximum number of UUIDs that can

 \* be inserted in the Solicited Services tag of AD

 \*/

#define ACI\_AD\_SOL\_SVCUUID\_128\_MAX\_COUNT 1

/\*\*

 \* @def ACI\_SEC\_ENCKEY\_SIZE\_MIN

 \* @brief Minimum encryption key size

 \*/

#define ACI\_SEC\_ENCKEY\_SIZE\_MIN 7

/\*\*

 \* @def ACI\_SEC\_ENCKEY\_SIZE\_MAX

 \* @brief Maximum encryption key size

 \*/

#define ACI\_SEC\_ENCKEY\_SIZE\_MAX 16

/\*\*

 \* @def ACI\_CUSTOM\_AD\_TYPE\_MAX\_COUNT

 \* @brief Maximum number of custom ad types

 \*/

#define ACI\_CUSTOM\_AD\_TYPE\_MAX\_COUNT 8

/\*\*

 \* @def ACI\_CUSTOM\_AD\_TYPE\_MAX\_DATA\_LENGTH

 \* @brief Maximum custom ad type data size

 \*/

#define ACI\_CUSTOM\_AD\_TYPE\_MAX\_DATA\_LENGTH 20

/\*\*

 \* @struct aci\_tx\_data\_t

 \* @brief Generic ACI transmit data structure

 \*/

typedef struct

{

 uint8\_t pipe\_number;

 uint8\_t aci\_data[ACI\_PIPE\_TX\_DATA\_MAX\_LEN];

} \_aci\_packed\_ aci\_tx\_data\_t;

ACI\_ASSERT\_SIZE(aci\_tx\_data\_t, ACI\_PIPE\_TX\_DATA\_MAX\_LEN + 1);

/\*\*

 \* @struct aci\_rx\_data\_t

 \* @brief Generic ACI receive data structure

 \*/

typedef struct

{

 uint8\_t pipe\_number;

 uint8\_t aci\_data[ACI\_PIPE\_RX\_DATA\_MAX\_LEN];

} \_aci\_packed\_ aci\_rx\_data\_t;

ACI\_ASSERT\_SIZE(aci\_rx\_data\_t, ACI\_PIPE\_RX\_DATA\_MAX\_LEN + 1);

/\*\*

 \* @enum aci\_hw\_error\_t

 \* @brief Hardware Error codes

 \*/

typedef enum

{

 ACI\_HW\_ERROR\_NONE = 0x00,

 ACI\_HW\_ERROR\_FATAL = 0x01

} \_aci\_packed\_ aci\_hw\_error\_t;

/\*\*

 \* @enum aci\_clock\_accuracy\_t

 \* @brief Bluetooth Low Energy Clock Accuracy

 \*/

typedef enum

{

 ACI\_CLOCK\_ACCURACY\_500\_PPM = 0x00,

 ACI\_CLOCK\_ACCURACY\_250\_PPM = 0x01,

 ACI\_CLOCK\_ACCURACY\_150\_PPM = 0x02,

 ACI\_CLOCK\_ACCURACY\_100\_PPM = 0x03,

 ACI\_CLOCK\_ACCURACY\_75\_PPM = 0x04,

 ACI\_CLOCK\_ACCURACY\_50\_PPM = 0x05,

 ACI\_CLOCK\_ACCURACY\_30\_PPM = 0x06,

 ACI\_CLOCK\_ACCURACY\_20\_PPM = 0x07

} \_aci\_packed\_ aci\_clock\_accuracy\_t;

/\*\*

 \* @enum aci\_app\_latency\_mode\_t

 \* @brief Application latency modes

 \*/

typedef enum

{

 ACI\_APP\_LATENCY\_DISABLE = 0,

 ACI\_APP\_LATENCY\_ENABLE = 1

} \_aci\_packed\_ aci\_app\_latency\_mode\_t;

/\*\*

 \* @enum gatt\_format\_t

 \* @brief GATT format definitions

 \*/

typedef enum

{

 ACI\_GATT\_FORMAT\_NONE = 0x00, /\*\*< No characteristic format available \*/

 ACI\_GATT\_FORMAT\_BOOLEAN = 0x01, /\*\*< Not Supported \*/

 ACI\_GATT\_FORMAT\_2BIT = 0x02, /\*\*< Not Supported \*/

 ACI\_GATT\_FORMAT\_NIBBLE = 0x03, /\*\*< Not Supported \*/

 ACI\_GATT\_FORMAT\_UINT8 = 0x04,

 ACI\_GATT\_FORMAT\_UINT12 = 0x05,

 ACI\_GATT\_FORMAT\_UINT16 = 0x06,

 ACI\_GATT\_FORMAT\_UINT24 = 0x07,

 ACI\_GATT\_FORMAT\_UINT32 = 0x08,

 ACI\_GATT\_FORMAT\_UINT48 = 0x09,

 ACI\_GATT\_FORMAT\_UINT64 = 0x0A,

 ACI\_GATT\_FORMAT\_UINT128 = 0x0B,

 ACI\_GATT\_FORMAT\_SINT8 = 0x0C,

 ACI\_GATT\_FORMAT\_SINT12 = 0x0D,

 ACI\_GATT\_FORMAT\_SINT16 = 0x0E,

 ACI\_GATT\_FORMAT\_SINT24 = 0x0F,

 ACI\_GATT\_FORMAT\_SINT32 = 0x10,

 ACI\_GATT\_FORMAT\_SINT48 = 0x11,

 ACI\_GATT\_FORMAT\_SINT64 = 0x12,

 ACI\_GATT\_FORMAT\_SINT128 = 0x13,

 ACI\_GATT\_FORMAT\_FLOAT32 = 0x14,

 ACI\_GATT\_FORMAT\_FLOAT64 = 0x15,

 ACI\_GATT\_FORMAT\_SFLOAT = 0x16,

 ACI\_GATT\_FORMAT\_FLOAT = 0x17,

 ACI\_GATT\_FORMAT\_DUINT16 = 0x18,

 ACI\_GATT\_FORMAT\_UTF8S = 0x19,

 ACI\_GATT\_FORMAT\_UTF16S = 0x1A,

 ACI\_GATT\_FORMAT\_STRUCT = 0x1B

} \_aci\_packed\_ aci\_gatt\_format\_t;

/\*\*

 \* @brief GATT Bluetooth namespace

 \*/

typedef enum

{

 ACI\_GATT\_NAMESPACE\_INVALID = 0x00,

 ACI\_GATT\_NAMESPACE\_BTSIG = 0x01 /\*\*< Bluetooth SIG \*/

} \_aci\_packed\_ aci\_gatt\_namespace\_t;

/\*\*

 \* @brief Security key types

 \*/

typedef enum

{

 ACI\_KEY\_TYPE\_INVALID = 0x00,

 ACI\_KEY\_TYPE\_PASSKEY = 0x01

} \_aci\_packed\_ aci\_key\_type\_t;

/\*\*

 \* @enum aci\_bond\_status\_code\_t

 \* @brief Bond status code

 \*/

typedef enum

{

 /\*\*

 \* Bonding succeeded

 \*/

 ACI\_BOND\_STATUS\_SUCCESS = 0x00,

 /\*\*

 \* Bonding failed

 \*/

 ACI\_BOND\_STATUS\_FAILED = 0x01,

 /\*\*

 \* Bonding error: Timeout can occur when link termination is unexpected or did not get connected OR SMP timer expired

 \*/

 ACI\_BOND\_STATUS\_FAILED\_TIMED\_OUT = 0x02,

 /\*\*

 \* Bonding error: Passkey entry failed

 \*/

 ACI\_BOND\_STATUS\_FAILED\_PASSKEY\_ENTRY\_FAILED = 0x81,

 /\*\*

 \* Bonding error: OOB unavailable

 \*/

 ACI\_BOND\_STATUS\_FAILED\_OOB\_UNAVAILABLE = 0x82,

 /\*\*

 \* Bonding error: Authentication request failed

 \*/

 ACI\_BOND\_STATUS\_FAILED\_AUTHENTICATION\_REQ = 0x83,

 /\*\*

 \* Bonding error: Confirm value failed

 \*/

 ACI\_BOND\_STATUS\_FAILED\_CONFIRM\_VALUE = 0x84,

 /\*\*

 \* Bonding error: Pairing unsupported

 \*/

 ACI\_BOND\_STATUS\_FAILED\_PAIRING\_UNSUPPORTED = 0x85,

 /\*\*

 \* Bonding error: Invalid encryption key size

 \*/

 ACI\_BOND\_STATUS\_FAILED\_ENCRYPTION\_KEY\_SIZE = 0x86,

 /\*\*

 \* Bonding error: Unsupported SMP command

 \*/

 ACI\_BOND\_STATUS\_FAILED\_SMP\_CMD\_UNSUPPORTED = 0x87,

 /\*\*

 \* Bonding error: Unspecified reason

 \*/

 ACI\_BOND\_STATUS\_FAILED\_UNSPECIFIED\_REASON = 0x88,

 /\*\*

 \* Bonding error: Too many attempts

 \*/

 ACI\_BOND\_STATUS\_FAILED\_REPEATED\_ATTEMPTS = 0x89,

 /\*\*

 \* Bonding error: Invalid parameters

 \*/

 ACI\_BOND\_STATUS\_FAILED\_INVALID\_PARAMETERS = 0x8A

} \_aci\_packed\_ aci\_bond\_status\_code\_t;

ACI\_ASSERT\_SIZE(aci\_bond\_status\_code\_t, 1);

/\*\*

 \* @enum aci\_bond\_status\_source\_t

 \* @brief Source of a bond status code

 \*/

typedef enum

{

 ACI\_BOND\_STATUS\_SOURCE\_INVALID = 0x00,

 ACI\_BOND\_STATUS\_SOURCE\_LOCAL = 0x01,

 ACI\_BOND\_STATUS\_SOURCE\_REMOTE = 0x02

} \_aci\_packed\_ aci\_bond\_status\_source\_t;

/\*\*

 \* @enum aci\_status\_code\_t

 \* @brief ACI status codes

 \*/

typedef enum

{

 /\*\*

 \* Success

 \*/

 ACI\_STATUS\_SUCCESS = 0x00,

 /\*\*

 \* Transaction continuation status

 \*/

 ACI\_STATUS\_TRANSACTION\_CONTINUE = 0x01,

 /\*\*

 \* Transaction completed

 \*/

 ACI\_STATUS\_TRANSACTION\_COMPLETE = 0x02,

 /\*\*

 \* Extended status, further checks needed

 \*/

 ACI\_STATUS\_EXTENDED = 0x03,

 /\*\*

 \* Unknown error.

 \*/

 ACI\_STATUS\_ERROR\_UNKNOWN = 0x80,

 /\*\*

 \* Internal error.

 \*/

 ACI\_STATUS\_ERROR\_INTERNAL = 0x81,

 /\*\*

 \* Unknown command

 \*/

 ACI\_STATUS\_ERROR\_CMD\_UNKNOWN = 0x82,

 /\*\*

 \* Command invalid in the current device state

 \*/

 ACI\_STATUS\_ERROR\_DEVICE\_STATE\_INVALID = 0x83,

 /\*\*

 \* Invalid length

 \*/

 ACI\_STATUS\_ERROR\_INVALID\_LENGTH = 0x84,

 /\*\*

 \* Invalid input parameters

 \*/

 ACI\_STATUS\_ERROR\_INVALID\_PARAMETER = 0x85,

 /\*\*

 \* Busy

 \*/

 ACI\_STATUS\_ERROR\_BUSY = 0x86,

 /\*\*

 \* Invalid data format or contents

 \*/

 ACI\_STATUS\_ERROR\_INVALID\_DATA = 0x87,

 /\*\*

 \* CRC mismatch

 \*/

 ACI\_STATUS\_ERROR\_CRC\_MISMATCH = 0x88,

 /\*\*

 \* Unsupported setup format

 \*/

 ACI\_STATUS\_ERROR\_UNSUPPORTED\_SETUP\_FORMAT = 0x89,

 /\*\*

 \* Invalid sequence number during a write dynamic data sequence

 \*/

 ACI\_STATUS\_ERROR\_INVALID\_SEQ\_NO = 0x8A,

 /\*\*

 \* Setup data is locked and cannot be modified

 \*/

 ACI\_STATUS\_ERROR\_SETUP\_LOCKED = 0x8B,

 /\*\*

 \* Setup error due to lock verification failure

 \*/

 ACI\_STATUS\_ERROR\_LOCK\_FAILED = 0x8C,

 /\*\*

 \* Bond required: Local Pipes need bonded/trusted peer

 \*/

 ACI\_STATUS\_ERROR\_BOND\_REQUIRED = 0x8D,

 /\*\*

 \* Command rejected as a transaction is still pending

 \*/

 ACI\_STATUS\_ERROR\_REJECTED = 0x8E,

 /\*\*

 \* Pipe Error Event : Data size exceeds size specified for pipe : Transmit failed

 \*/

 ACI\_STATUS\_ERROR\_DATA\_SIZE = 0x8F,

 /\*\*

 \* Pipe Error Event : Invalid pipe

 \*/

 ACI\_STATUS\_ERROR\_PIPE\_INVALID = 0x90,

 /\*\*

 \* Pipe Error Event : Credit not available

 \*/

 ACI\_STATUS\_ERROR\_CREDIT\_NOT\_AVAILABLE = 0x91,

 /\*\*

 \* Pipe Error Event : Peer device has sent an error on an pipe operation on the remote characteristic

 \*/

 ACI\_STATUS\_ERROR\_PEER\_ATT\_ERROR = 0x92,

 /\*\*

 \* Connection was not established before the BTLE advertising was stopped

 \*/

 ACI\_STATUS\_ERROR\_ADVT\_TIMEOUT = 0x93,

 /\*\*

 \* Peer has triggered a Security Manager Protocol Error

 \*/

 ACI\_STATUS\_ERROR\_PEER\_SMP\_ERROR = 0x94,

 /\*\*

 \* Pipe Error Event : Pipe type invalid for the selected operation

 \*/

 ACI\_STATUS\_ERROR\_PIPE\_TYPE\_INVALID = 0x95,

 /\*\*

 \* Pipe Error Event : Pipe state invalid for the selected operation

 \*/

 ACI\_STATUS\_ERROR\_PIPE\_STATE\_INVALID = 0x96,

 /\*\*

 \* Invalid key size provided

 \*/

 ACI\_STATUS\_ERROR\_INVALID\_KEY\_SIZE = 0x97,

 /\*\*

 \* Invalid key data provided

 \*/

 ACI\_STATUS\_ERROR\_INVALID\_KEY\_DATA = 0x98,

 /\*\*

 \* Reserved range start

 \*/

 ACI\_STATUS\_RESERVED\_START = 0xF0,

 /\*\*

 \* Reserved range end

 \*/

 ACI\_STATUS\_RESERVED\_END = 0xFF

} \_aci\_packed\_ aci\_status\_code\_t;

ACI\_ASSERT\_SIZE(aci\_status\_code\_t, 1);

/\*\*

 \* @}

 \*/

#endif // ACI\_H\_\_

2. aci\_cmds.h

#ifndef ACI\_CMDS\_H\_\_

#define ACI\_CMDS\_H\_\_

#include "aci.h"

/\*\*

 \* @enum aci\_cmd\_opcode\_t

 \* @brief ACI command opcodes

 \*/

typedef enum

{

 /\*\*

 \* Enter test mode

 \*/

 ACI\_CMD\_TEST = 0x01,

 /\*\*

 \* Echo (loopback) test command

 \*/

 ACI\_CMD\_ECHO = 0x02,

 /\*\*

 \* Send a BTLE DTM command to the radio

 \*/

 ACI\_CMD\_DTM\_CMD = 0x03,

 /\*\*

 \* Put the device to sleep

 \*/

 ACI\_CMD\_SLEEP = 0x04,

 /\*\*

 \* Wakeup the device from deep sleep

 \*/

 ACI\_CMD\_WAKEUP = 0x05,

 /\*\*

 \* Replace the contents of the internal database with

 \* user provided data

 \*/

 ACI\_CMD\_SETUP = 0x06,

 /\*\*

 \* Read the portions of memory required to be restored after a power cycle

 \*/

 ACI\_CMD\_READ\_DYNAMIC\_DATA = 0x07,

 /\*\*

 \* Write back the data retrieved using ACI\_CMD\_READ\_DYNAMIC\_DATA

 \*/

 ACI\_CMD\_WRITE\_DYNAMIC\_DATA = 0x08,

 /\*\*

 \* Retrieve the device's version information

 \*/

 ACI\_CMD\_GET\_DEVICE\_VERSION = 0x09,

 /\*\*

 \* Request the Bluetooth address and its type

 \*/

 ACI\_CMD\_GET\_DEVICE\_ADDRESS = 0x0A,

 /\*\*

 \* Request the battery level measured by nRF8001

 \*/

 ACI\_CMD\_GET\_BATTERY\_LEVEL = 0x0B,

 /\*\*

 \* Request the temperature value measured by nRF8001

 \*/

 ACI\_CMD\_GET\_TEMPERATURE = 0x0C,

 /\*\*

 \* Write to the local Attribute Database

 \*/

 ACI\_CMD\_SET\_LOCAL\_DATA = 0x0D,

 /\*\*

 \* Reset the baseband and radio and go back to idle

 \*/

 ACI\_CMD\_RADIO\_RESET = 0x0E,

 /\*\*

 \* Start advertising and wait for a master connection

 \*/

 ACI\_CMD\_CONNECT = 0x0F,

 /\*\*

 \* Start advertising and wait for a master connection

 \*/

 ACI\_CMD\_BOND = 0x10,

 /\*\*

 \* Start advertising and wait for a master connection

 \*/

 ACI\_CMD\_DISCONNECT = 0x11,

 /\*\*

 \* Throttles the Radio transmit power

 \*/

 ACI\_CMD\_SET\_TX\_POWER = 0x12,

 /\*\*

 \* Trigger a connection parameter update

 \*/

 ACI\_CMD\_CHANGE\_TIMING = 0x13,

 /\*\*

 \* Open a remote pipe for data reception

 \*/

 ACI\_CMD\_OPEN\_REMOTE\_PIPE = 0x14,

 /\*\*

 \* Transmit data over an open pipe

 \*/

 ACI\_CMD\_SEND\_DATA = 0x15,

 /\*\*

 \* Send an acknowledgment of received data

 \*/

 ACI\_CMD\_SEND\_DATA\_ACK = 0x16,

 /\*\*

 \* Request data over an open pipe

 \*/

 ACI\_CMD\_REQUEST\_DATA = 0x17,

 /\*\*

 \* NACK a data reception

 \*/

 ACI\_CMD\_SEND\_DATA\_NACK = 0x18,

 /\*\*

 \* Set application latency

 \*/

 ACI\_CMD\_SET\_APP\_LATENCY = 0x19,

 /\*\*

 \* Set a security key

 \*/

 ACI\_CMD\_SET\_KEY = 0x1A,

 /\*\*

 \* Open Advertising Pipes

 \*/

 ACI\_CMD\_OPEN\_ADV\_PIPE = 0x1B,

 /\*\*

 \* Start non-connectable advertising

 \*/

 ACI\_CMD\_BROADCAST = 0x1C,

 /\*\*

 \* Start a security request in bonding mode

 \*/

 ACI\_CMD\_BOND\_SECURITY\_REQUEST = 0x1D,

 /\*\*

 \* Start Directed advertising towards a Bonded Peer

 \*/

 ACI\_CMD\_CONNECT\_DIRECT = 0x1E,

 /\*\*

 \* Close a previously opened remote pipe

 \*/

 ACI\_CMD\_CLOSE\_REMOTE\_PIPE = 0x1F,

 /\*\*

 \* Invalid ACI command opcode

 \*/

 ACI\_CMD\_INVALID = 0xFF

} \_aci\_packed\_ aci\_cmd\_opcode\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_opcode\_t, 1);

/\*\*

 \* @struct aci\_cmd\_params\_test\_t

 \* @brief Structure for the ACI\_CMD\_TEST ACI command parameters

 \*/

typedef struct

{

 aci\_test\_mode\_change\_t test\_mode\_change; /\*\*< enum aci\_test\_mode\_change\_t \*/

} \_aci\_packed\_ aci\_cmd\_params\_test\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_test\_t, 1);

/\*\*

 \* @struct aci\_cmd\_params\_echo\_t

 \* @brief Structure for the ACI\_CMD\_ECHO ACI command parameters

 \*/

typedef struct

{

 uint8\_t echo\_data[ACI\_ECHO\_DATA\_MAX\_LEN];

} \_aci\_packed\_ aci\_cmd\_params\_echo\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_echo\_t, ACI\_ECHO\_DATA\_MAX\_LEN);

/\*\*

 \* @struct aci\_cmd\_params\_dtm\_cmd\_t

 \* @brief Structure for the ACI\_CMD\_DTM\_CMD ACI command parameters

 \*/

typedef struct

{

 uint8\_t cmd\_msb;

 uint8\_t cmd\_lsb;

} \_aci\_packed\_ aci\_cmd\_params\_dtm\_cmd\_t;

/\*\*

 \* @struct aci\_cmd\_params\_setup\_t

 \* @brief Structure for the ACI\_CMD\_SETUP ACI command parameters

 \*/

typedef struct

{

 uint8\_t setup\_data[1];

} \_aci\_packed\_ aci\_cmd\_params\_setup\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_setup\_t, 1);

/\*\*

 \* @struct aci\_cmd\_params\_write\_dynamic\_data\_t

 \* @brief Structure for the ACI\_CMD\_WRITE\_DYNAMIC\_DATA ACI command parameters

 \* @note Dynamic data chunk size in this command is defined to go up to ACI\_PACKET\_MAX\_LEN - 3

 \*/

typedef struct

{

 uint8\_t seq\_no;

 uint8\_t dynamic\_data[1];

} \_aci\_packed\_ aci\_cmd\_params\_write\_dynamic\_data\_t;

/\*\*

 \* @define aci\_cmd\_params\_set\_local\_data\_t

 \* @brief Structure for the ACI\_CMD\_SET\_LOCAL\_DATA ACI command parameters

 \*/

typedef struct

{

 aci\_tx\_data\_t tx\_data;

} \_aci\_packed\_ aci\_cmd\_params\_set\_local\_data\_t;

/\*\*

 \* @struct aci\_cmd\_params\_connect\_t

 \* @brief Structure for the ACI\_CMD\_CONNECT ACI command parameters

 \*/

typedef struct

{

 uint16\_t timeout; /\*\*< 0x0000 (no timeout) to 0x3FFF \*/

 uint16\_t adv\_interval; /\*\*< 16 bits of advertising interval for general discovery \*/

} \_aci\_packed\_ aci\_cmd\_params\_connect\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_connect\_t, 4);

/\*\*

 \* @define aci\_cmd\_params\_bond\_t

 \* @brief Structure for the ACI\_CMD\_BOND ACI command parameters

 \*/

typedef struct

{

 uint16\_t timeout; /\*\*< 0x0000 (no timeout) to 0x3FFF \*/

 uint16\_t adv\_interval; /\*\*< 16 bits of advertising interval for general discovery \*/

} \_aci\_packed\_ aci\_cmd\_params\_bond\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_bond\_t, 4);

/\*\*

 \* @struct aci\_cmd\_params\_disconnect\_t

 \* @brief Structure for the ACI\_CMD\_DISCONNECT ACI command parameters

 \*/

typedef struct

{

 aci\_disconnect\_reason\_t reason; /\*\*< enum aci\_disconnect\_reason\_t \*/

} \_aci\_packed\_ aci\_cmd\_params\_disconnect\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_disconnect\_t, 1);

/\*\*

 \* @struct aci\_cmd\_params\_set\_tx\_power\_t

 \* @brief Structure for the ACI\_CMD\_SET\_TX\_POWER ACI command parameters

 \*/

typedef struct

{

 aci\_device\_output\_power\_t device\_power; /\*\*< enum aci\_device\_output\_power\_t \*/

} \_aci\_packed\_ aci\_cmd\_params\_set\_tx\_power\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_set\_tx\_power\_t, 1);

/\*\*

 \* @struct aci\_cmd\_params\_change\_timing\_t

 \* @brief Structure for the ACI\_CMD\_CHANGE\_TIMING ACI command parameters

 \*/

typedef struct

{

 aci\_ll\_conn\_params\_t conn\_params;

} \_aci\_packed\_ aci\_cmd\_params\_change\_timing\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_change\_timing\_t, 8);

/\*\*

 \* @struct aci\_cmd\_params\_open\_remote\_pipe\_t

 \* @brief Structure for the ACI\_CMD\_OPEN\_REMOTE\_PIPE ACI command parameters

 \*/

typedef struct

{

 uint8\_t pipe\_number;

} \_aci\_packed\_ aci\_cmd\_params\_open\_remote\_pipe\_t;

/\*\*

 \* @struct aci\_cmd\_params\_send\_data\_t

 \* @brief Structure for the ACI\_CMD\_SEND\_DATA ACI command parameters

 \*/

typedef struct

{

 aci\_tx\_data\_t tx\_data;

} \_aci\_packed\_ aci\_cmd\_params\_send\_data\_t;

/\*\*

 \* @define aci\_cmd\_params\_send\_data\_ack\_t

 \* @brief Structure for the ACI\_CMD\_SEND\_DATA\_ACK ACI command parameters

 \*/

typedef struct

{

 uint8\_t pipe\_number;

} \_aci\_packed\_ aci\_cmd\_params\_send\_data\_ack\_t;

/\*\*

 \* @struct aci\_cmd\_params\_send\_data\_t

 \* @brief Structure for the ACI\_CMD\_SEND\_DATA ACI command parameters

 \*/

typedef struct

{

 uint8\_t pipe\_number;

} \_aci\_packed\_ aci\_cmd\_params\_request\_data\_t;

/\*\*

 \* @define aci\_cmd\_params\_send\_data\_nack\_t

 \* @brief Structure for the ACI\_CMD\_SEND\_DATA\_NACK ACI command parameters

 \*/

typedef struct

{

 uint8\_t pipe\_number;

 uint8\_t error\_code;

} \_aci\_packed\_ aci\_cmd\_params\_send\_data\_nack\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_send\_data\_nack\_t, 2);

/\*\*

 \* @define aci\_cmd\_params\_set\_app\_latency\_t

 \* @brief Structure for the ACI\_CMD\_SET\_APP\_LATENCY ACI command parameters

 \*/

typedef struct

{

 aci\_app\_latency\_mode\_t mode;

 uint16\_t latency;

} \_aci\_packed\_ aci\_cmd\_params\_set\_app\_latency\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_set\_app\_latency\_t, 3);

/\*\*

 \* @define aci\_cmd\_params\_set\_key\_t

 \* @brief Structure for the ACI\_CMD\_SET\_KEY ACI command parameters

 \*/

typedef struct

{

 aci\_key\_type\_t key\_type;

 union

 {

 uint8\_t passkey[6];

 uint8\_t oob\_key[16];

 } key;

} \_aci\_packed\_ aci\_cmd\_params\_set\_key\_t;

ACI\_ASSERT\_SIZE(aci\_cmd\_params\_set\_key\_t, 17);

/\*\*

 \* @define aci\_cmd\_params\_open\_adv\_pipe\_t

 \* @brief Structure for the ACI\_CMD\_OPEN\_ADV\_PIPE ACI command parameters

 \*/

typedef struct

{

 uint8\_t pipes[8];

} \_aci\_packed\_ aci\_cmd\_params\_open\_adv\_pipe\_t;

/\*\*

 \* @define aci\_cmd\_params\_broadcast\_t

 \* @brief Structure for the ACI\_CMD\_BROADCAST ACI command parameters

 \*/

typedef struct

{

 uint16\_t timeout; /\*\*< 0x0000 (no timeout) to 0x3FFF \*/

 uint16\_t adv\_interval; /\*\*< 16 bits of advertising interval for general discovery \*/

} \_aci\_packed\_ aci\_cmd\_params\_broadcast\_t;

/\*\*

 \* @struct aci\_cmd\_params\_close\_remote\_pipe\_t

 \* @brief Structure for the ACI\_CMD\_CLOSE\_REMOTE\_PIPE ACI command parameters

 \*/

typedef struct

{

 uint8\_t pipe\_number;

} \_aci\_packed\_ aci\_cmd\_params\_close\_remote\_pipe\_t;

/\*\*

 \* @struct aci\_cmd\_t

 \* @brief Encapsulates a generic ACI command

 \*/

typedef struct

{

 uint8\_t len; /\*\*< Length of the ACI command \*/

 aci\_cmd\_opcode\_t cmd\_opcode; /\*\*< enum aci\_cmd\_opcode\_t -> Opcode of the ACI command \*/

 union

 {

 aci\_cmd\_params\_test\_t test;

 aci\_cmd\_params\_echo\_t echo;

 aci\_cmd\_params\_dtm\_cmd\_t dtm\_cmd;

 aci\_cmd\_params\_setup\_t setup;

 aci\_cmd\_params\_write\_dynamic\_data\_t write\_dynamic\_data;

 aci\_cmd\_params\_set\_local\_data\_t set\_local\_data;

 aci\_cmd\_params\_connect\_t connect;

 aci\_cmd\_params\_bond\_t bond;

 aci\_cmd\_params\_disconnect\_t disconnect;

 aci\_cmd\_params\_set\_tx\_power\_t set\_tx\_power;

 aci\_cmd\_params\_change\_timing\_t change\_timing;

 aci\_cmd\_params\_open\_remote\_pipe\_t open\_remote\_pipe;

 aci\_cmd\_params\_send\_data\_t send\_data;

 aci\_cmd\_params\_send\_data\_ack\_t send\_data\_ack;

 aci\_cmd\_params\_request\_data\_t request\_data;

 aci\_cmd\_params\_send\_data\_nack\_t send\_data\_nack;

 aci\_cmd\_params\_set\_app\_latency\_t set\_app\_latency;

 aci\_cmd\_params\_set\_key\_t set\_key;

 aci\_cmd\_params\_open\_adv\_pipe\_t open\_adv\_pipe;

 aci\_cmd\_params\_broadcast\_t broadcast;

 aci\_cmd\_params\_close\_remote\_pipe\_t close\_remote\_pipe;

 } params;

} \_aci\_packed\_ aci\_cmd\_t;

#endif // ACI\_CMDS\_H\_\_

3. aci\_events.h

#ifndef ACI\_EVTS\_H\_\_

#define ACI\_EVTS\_H\_\_

#include "aci.h"

/\*\*

 \* @enum aci\_evt\_opcode\_t

 \* @brief ACI event opcodes

 \*/

typedef enum

{

 /\*\*

 \* Invalid event code

 \*/

 ACI\_EVT\_INVALID = 0x00,

 /\*\*

 \* Sent every time the device starts

 \*/

 ACI\_EVT\_DEVICE\_STARTED = 0x81,

 /\*\*

 \* Mirrors the ACI\_CMD\_ECHO

 \*/

 ACI\_EVT\_ECHO = 0x82,

 /\*\*

 \* Asynchronous hardware error event

 \*/

 ACI\_EVT\_HW\_ERROR = 0x83,

 /\*\*

 \* Event opcode used as a event response for all commands

 \*/

 ACI\_EVT\_CMD\_RSP = 0x84,

 /\*\*

 \* Link connected

 \*/

 ACI\_EVT\_CONNECTED = 0x85,

 /\*\*

 \* Link disconnected

 \*/

 ACI\_EVT\_DISCONNECTED = 0x86,

 /\*\*

 \* Bond completion result

 \*/

 ACI\_EVT\_BOND\_STATUS = 0x87,

 /\*\*

 \* Pipe bitmap for available pipes

 \*/

 ACI\_EVT\_PIPE\_STATUS = 0x88,

 /\*\*

 \* Sent to the application when the radio enters a connected state

 \* or when the timing of the radio connection changes

 \*/

 ACI\_EVT\_TIMING = 0x89,

 /\*\*

 \* Notification to the application that transmit credits are

 \* available

 \*/

 ACI\_EVT\_DATA\_CREDIT = 0x8A,

 /\*\*

 \* Data acknowledgement event

 \*/

 ACI\_EVT\_DATA\_ACK = 0x8B,

 /\*\*

 \* Data received notification event

 \*/

 ACI\_EVT\_DATA\_RECEIVED = 0x8C,

 /\*\*

 \* Error notification event

 \*/

 ACI\_EVT\_PIPE\_ERROR = 0x8D,

 /\*\*

 \* Display Passkey Event

 \*/

 ACI\_EVT\_DISPLAY\_PASSKEY = 0x8E,

 /\*\*

 \* Security Key request

 \*/

 ACI\_EVT\_KEY\_REQUEST = 0x8F

} \_aci\_packed\_ aci\_evt\_opcode\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_opcode\_t, 1);

/\*\*

 \* @struct aci\_evt\_params\_device\_started\_t

 \* @brief Structure for the ACI\_EVT\_DEVICE\_STARTED event return parameters

 \*/

typedef struct

{

 aci\_device\_operation\_mode\_t device\_mode; /\*\*< Mode in which the device is being started \*/

 aci\_hw\_error\_t hw\_error; /\*\*< Hardware Error if available for the start \*/

 uint8\_t credit\_available; /\*\*< Flow control credit available for this specific FW build \*/

} \_aci\_packed\_ aci\_evt\_params\_device\_started\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_params\_device\_started\_t, 3);

/\*\*

 \* @struct aci\_evt\_params\_hw\_error\_t

 \* @brief Structure for the ACI\_EVT\_HW\_ERROR event return parameters

 \*/

typedef struct

{

 uint16\_t line\_num;

 uint8\_t file\_name[20];

} \_aci\_packed\_ aci\_evt\_params\_hw\_error\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_params\_hw\_error\_t, 22);

/\*\*

 \* @struct aci\_evt\_cmd\_rsp\_params\_dtm\_cmd\_t

 \* @brief Structure for the ACI\_EVT\_CMD\_RSP event with opcode=ACI\_CMD\_DTM\_CMD event return parameters

 \*/

typedef struct

{

 uint8\_t evt\_msb;

 uint8\_t evt\_lsb;

} \_aci\_packed\_ aci\_evt\_cmd\_rsp\_params\_dtm\_cmd\_t;

/\*\*

 \* @struct aci\_evt\_cmd\_rsp\_read\_dynamic\_data\_t

 \* @brief Structure for the ACI\_EVT\_CMD\_RSP event with opcode=ACI\_CMD\_READ\_DYNAMIC\_DATA event return parameters

 \* @note Dynamic data chunk size in this event is defined to go up to ACI\_PACKET\_MAX\_LEN - 5

 \*/

typedef struct

{

 uint8\_t seq\_no;

 uint8\_t dynamic\_data[1];

} \_aci\_packed\_ aci\_evt\_cmd\_rsp\_read\_dynamic\_data\_t;

/\*\*

 \* @struct aci\_evt\_cmd\_rsp\_params\_get\_device\_version\_t

 \* @brief Structure for the ACI\_EVT\_CMD\_RSP event with opcode=ACI\_CMD\_GET\_DEVICE\_VERSION event return parameters

 \*/

typedef struct

{

 uint16\_t configuration\_id;

 uint8\_t aci\_version;

 uint8\_t setup\_format;

 uint32\_t setup\_id;

 uint8\_t setup\_status;

} \_aci\_packed\_ aci\_evt\_cmd\_rsp\_params\_get\_device\_version\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_cmd\_rsp\_params\_get\_device\_version\_t, 9);

/\*\*

 \* @struct aci\_evt\_cmd\_rsp\_params\_get\_device\_address\_t

 \* @brief Structure for the ACI\_EVT\_CMD\_RSP event with opcode=ACI\_CMD\_GET\_DEVICE\_ADDRESS event return parameters

 \*/

typedef struct

{

 uint8\_t bd\_addr\_own[BTLE\_DEVICE\_ADDRESS\_SIZE];

 aci\_bd\_addr\_type\_t bd\_addr\_type;

} \_aci\_packed\_ aci\_evt\_cmd\_rsp\_params\_get\_device\_address\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_cmd\_rsp\_params\_get\_device\_address\_t, BTLE\_DEVICE\_ADDRESS\_SIZE + 1);

/\*\*

 \* @struct aci\_evt\_cmd\_rsp\_params\_get\_battery\_level\_t

 \* @brief Structure for the ACI\_EVT\_CMD\_RSP event with opcode=ACI\_CMD\_GET\_BATTERY\_LEVEL event return parameters

 \*/

typedef struct

{

 uint16\_t battery\_level;

} \_aci\_packed\_ aci\_evt\_cmd\_rsp\_params\_get\_battery\_level\_t;

/\*\*

 \* @struct aci\_evt\_cmd\_rsp\_params\_get\_temperature\_t

 \* @brief Structure for the ACI\_EVT\_CMD\_RSP event with opcode=ACI\_CMD\_GET\_TEMPERATURE event return parameters

 \*/

typedef struct

{

 int16\_t temperature\_value;

} \_aci\_packed\_ aci\_evt\_cmd\_rsp\_params\_get\_temperature\_t;

/\*\*

 \* @struct aci\_evt\_params\_cmd\_rsp\_t

 \* @brief Structure for the ACI\_EVT\_CMD\_RSP event return parameters

 \*/

typedef struct

{

 aci\_cmd\_opcode\_t cmd\_opcode; /\*\*< Command opcode for which the event response is being sent \*/

 aci\_status\_code\_t cmd\_status; /\*\*< Status of the command that was sent. Used in the context of the command. \*/

 union

 {

 aci\_evt\_cmd\_rsp\_params\_dtm\_cmd\_t dtm\_cmd;

 aci\_evt\_cmd\_rsp\_read\_dynamic\_data\_t read\_dynamic\_data;

 aci\_evt\_cmd\_rsp\_params\_get\_device\_version\_t get\_device\_version;

 aci\_evt\_cmd\_rsp\_params\_get\_device\_address\_t get\_device\_address;

 aci\_evt\_cmd\_rsp\_params\_get\_battery\_level\_t get\_battery\_level;

 aci\_evt\_cmd\_rsp\_params\_get\_temperature\_t get\_temperature;

 uint8\_t padding[29];

 } params;

} \_aci\_packed\_ aci\_evt\_params\_cmd\_rsp\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_params\_cmd\_rsp\_t, 31);

/\*\*

 \* @struct aci\_evt\_params\_connected\_t

 \* @brief Structure for the ACI\_EVT\_CONNECTED event return parameters

 \*/

typedef struct

{

 aci\_bd\_addr\_type\_t dev\_addr\_type;

 uint8\_t dev\_addr[BTLE\_DEVICE\_ADDRESS\_SIZE];

 uint16\_t conn\_rf\_interval; /\*\*< rf\_interval = conn\_rf\_interval \* 1.25 ms Range:0x0006 to 0x0C80 \*/

 uint16\_t conn\_slave\_rf\_latency; /\*\*< Number of RF events the slave can skip \*/

 uint16\_t conn\_rf\_timeout; /\*\*< Timeout as a multiple of 10ms i.e timeout = conn\_rf\_timeout \* 10ms Range: 0x000A to 0x0C80 \*/

 aci\_clock\_accuracy\_t master\_clock\_accuracy; /\*\*< Clock accuracy of Bluetooth master: Enumerated list of values from 500 ppm to 20 ppm \*/

} \_aci\_packed\_ aci\_evt\_params\_connected\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_params\_connected\_t, 14);

/\*\*

 \* @struct aci\_evt\_params\_disconnected\_t

 \* @brief Structure for the ACI\_EVT\_DISCONNECTED event return parameters

 \*/

typedef struct

{

 aci\_status\_code\_t aci\_status;

 uint8\_t btle\_status;

} \_aci\_packed\_ aci\_evt\_params\_disconnected\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_params\_disconnected\_t, 2);

/\*\*

 \* @struct aci\_evt\_params\_bond\_status\_t

 \* @brief Structure for the ACI\_EVT\_BOND\_STATUS event return parameters

 \*/

typedef struct

{

 aci\_bond\_status\_code\_t status\_code;

 aci\_bond\_status\_source\_t status\_source;

 uint8\_t secmode1\_bitmap;

 uint8\_t secmode2\_bitmap;

 uint8\_t keys\_exchanged\_slave;

 uint8\_t keys\_exchanged\_master;

} \_aci\_packed\_ aci\_evt\_params\_bond\_status\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_params\_bond\_status\_t, 6);

/\*\*

 \* @struct aci\_evt\_params\_pipe\_status\_t

 \* @brief Structure for the ACI\_EVT\_PIPE\_STATUS event return parameters

 \*/

typedef struct

{

 uint8\_t pipes\_open\_bitmap[8];

 uint8\_t pipes\_closed\_bitmap[8];

} \_aci\_packed\_ aci\_evt\_params\_pipe\_status\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_params\_pipe\_status\_t, 16);

/\*\*

 \* @struct aci\_evt\_params\_timing\_t

 \* @brief Structure for the ACI\_EVT\_TIMING event return parameters

 \*/

typedef struct

{

 uint16\_t conn\_rf\_interval; /\*\*< rf\_interval = conn\_rf\_interval \* 1.25 ms Range:0x0006 to 0x0C80 \*/

 uint16\_t conn\_slave\_rf\_latency; /\*\*< Number of RF events the slave can skip \*/

 uint16\_t conn\_rf\_timeout; /\*\*< Timeout as a multiple of 10ms i.e timeout = conn\_rf\_timeout \* 10ms Range: 0x000A to 0x0C80 \*/

} \_aci\_packed\_ aci\_evt\_params\_timing\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_params\_timing\_t, 6);

/\*\*

 \* @struct aci\_evt\_params\_data\_credit\_t

 \* @brief Structure for the ACI\_EVT\_DATA\_CREDIT event return parameters

 \*/

typedef struct

{

 uint8\_t credit;

} \_aci\_packed\_ aci\_evt\_params\_data\_credit\_t;

/\*\*

 \* @struct aci\_evt\_params\_data\_ack\_t

 \* @brief Structure for the ACI\_EVT\_DATA\_ACK event return parameters

 \*/

typedef struct

{

 uint8\_t pipe\_number;

} \_aci\_packed\_ aci\_evt\_params\_data\_ack\_t;

/\*\*

 \* @struct aci\_evt\_params\_data\_received\_t

 \* @brief Structure for the ACI\_EVT\_DATA\_RECEIVED event return parameters

 \*/

typedef struct

{

 aci\_rx\_data\_t rx\_data;

} \_aci\_packed\_ aci\_evt\_params\_data\_received\_t;

typedef struct

{

 uint8\_t content[1];

} \_aci\_packed\_ error\_data\_t;

/\*\*

 \* @struct aci\_evt\_params\_pipe\_error\_t

 \* @brief Structure for the ACI\_EVT\_PIPE\_ERROR event return parameters

 \*/

typedef struct

{

 uint8\_t pipe\_number;

 uint8\_t error\_code;

 union

 {

 error\_data\_t error\_data;

 } params;

} \_aci\_packed\_ aci\_evt\_params\_pipe\_error\_t;

/\*\*

 \* @struct aci\_evt\_params\_display\_passkey\_t

 \* @brief Structure for the ACI\_EVT\_DISPLAY\_PASSKEY event return parameters

 \*/

typedef struct

{

 uint8\_t passkey[6];

} \_aci\_packed\_ aci\_evt\_params\_display\_passkey\_t;

/\*\*

 \* @struct aci\_evt\_params\_key\_request\_t

 \* @brief Structure for the ACI\_EVT\_KEY\_REQUEST event return parameters

 \*/

typedef struct

{

 aci\_key\_type\_t key\_type;

} \_aci\_packed\_ aci\_evt\_params\_key\_request\_t;

/\*\*

 \* @struct aci\_event\_params\_echo\_t

 \* @brief Structure for the ACI\_EVT\_ECHO ACI event parameters

 \*/

typedef struct

{

 uint8\_t echo\_data[ACI\_ECHO\_DATA\_MAX\_LEN];

} \_aci\_packed\_ aci\_evt\_params\_echo\_t;

/\*\*

 \* @struct aci\_evt\_t

 \* @brief Encapsulates a generic ACI event

 \*/

typedef struct

{

 uint8\_t len;

 aci\_evt\_opcode\_t evt\_opcode;

 union

 {

 aci\_evt\_params\_device\_started\_t device\_started;

 aci\_evt\_params\_echo\_t echo;

 aci\_evt\_params\_hw\_error\_t hw\_error;

 aci\_evt\_params\_cmd\_rsp\_t cmd\_rsp;

 aci\_evt\_params\_connected\_t connected;

 aci\_evt\_params\_disconnected\_t disconnected;

 aci\_evt\_params\_bond\_status\_t bond\_status;

 aci\_evt\_params\_pipe\_status\_t pipe\_status;

 aci\_evt\_params\_timing\_t timing;

 aci\_evt\_params\_data\_credit\_t data\_credit;

 aci\_evt\_params\_data\_ack\_t data\_ack;

 aci\_evt\_params\_data\_received\_t data\_received;

 aci\_evt\_params\_pipe\_error\_t pipe\_error;

 aci\_evt\_params\_display\_passkey\_t display\_passkey;

 aci\_evt\_params\_key\_request\_t key\_request;

 } params;

} \_aci\_packed\_ aci\_evt\_t;

ACI\_ASSERT\_SIZE(aci\_evt\_t, 33);

#endif // ACI\_EVTS\_H\_\_

4. aci\_protocol\_defines.h

#ifndef ACI\_OFFSET\_H\_\_

#define ACI\_OFFSET\_H\_\_

 #define OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_MIN\_CONN\_INTERVAL\_LSB 0

 #define OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_MIN\_CONN\_INTERVAL\_MSB 1

 #define OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_MAX\_CONN\_INTERVAL\_LSB 2

 #define OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_MAX\_CONN\_INTERVAL\_MSB 3

 #define OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_SLAVE\_LATENCY\_LSB 4

 #define OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_SLAVE\_LATENCY\_MSB 5

 #define OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_TIMEOUT\_MULT\_LSB 6

 #define OFFSET\_ACI\_LL\_CONN\_PARAMS\_T\_TIMEOUT\_MULT\_MSB 7

 #define OFFSET\_ACI\_TX\_DATA\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_TX\_DATA\_T\_ACI\_DATA 1

 #define OFFSET\_ACI\_RX\_DATA\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_RX\_DATA\_T\_ACI\_DATA 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_TEST\_T\_TEST\_MODE\_CHANGE 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_ECHO\_T\_ECHO\_DATA 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_DTM\_CMD\_T\_CMD\_MSB 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_DTM\_CMD\_T\_CMD\_LSB 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_SETUP\_T\_SETUP\_DATA 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_WRITE\_DYNAMIC\_DATA\_T\_SEQ\_NO 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_WRITE\_DYNAMIC\_DATA\_T\_DYNAMIC\_DATA 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_SET\_LOCAL\_DATA\_T\_TX\_DATA 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_CONNECT\_T\_TIMEOUT\_LSB 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_CONNECT\_T\_TIMEOUT\_MSB 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_CONNECT\_T\_ADV\_INTERVAL\_LSB 2

 #define OFFSET\_ACI\_CMD\_PARAMS\_CONNECT\_T\_ADV\_INTERVAL\_MSB 3

 #define OFFSET\_ACI\_CMD\_PARAMS\_BOND\_T\_TIMEOUT\_LSB 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_BOND\_T\_TIMEOUT\_MSB 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_BOND\_T\_ADV\_INTERVAL\_LSB 2

 #define OFFSET\_ACI\_CMD\_PARAMS\_BOND\_T\_ADV\_INTERVAL\_MSB 3

 #define OFFSET\_ACI\_CMD\_PARAMS\_DISCONNECT\_T\_REASON 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_SET\_TX\_POWER\_T\_DEVICE\_POWER 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_CHANGE\_TIMING\_T\_CONN\_PARAMS 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_OPEN\_REMOTE\_PIPE\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_T\_TX\_DATA 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_ACK\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_REQUEST\_DATA\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_NACK\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_SEND\_DATA\_NACK\_T\_ERROR\_CODE 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_SET\_APP\_LATENCY\_T\_MODE 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_SET\_APP\_LATENCY\_T\_LATENCY\_LSB 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_SET\_APP\_LATENCY\_T\_LATENCY\_MSB 2

 #define OFFSET\_ACI\_CMD\_PARAMS\_SET\_KEY\_T\_KEY\_TYPE 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_SET\_KEY\_T\_PASSKEY 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_SET\_KEY\_T\_OOB\_KEY 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_OPEN\_ADV\_PIPE\_T\_PIPES 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_BROADCAST\_T\_TIMEOUT\_LSB 0

 #define OFFSET\_ACI\_CMD\_PARAMS\_BROADCAST\_T\_TIMEOUT\_MSB 1

 #define OFFSET\_ACI\_CMD\_PARAMS\_BROADCAST\_T\_ADV\_INTERVAL\_LSB 2

 #define OFFSET\_ACI\_CMD\_PARAMS\_BROADCAST\_T\_ADV\_INTERVAL\_MSB 3

 #define OFFSET\_ACI\_CMD\_PARAMS\_CLOSE\_REMOTE\_PIPE\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_CMD\_T\_LEN 0

 #define OFFSET\_ACI\_CMD\_T\_CMD\_OPCODE 1

 #define OFFSET\_ACI\_CMD\_T\_TEST 2

 #define OFFSET\_ACI\_CMD\_T\_ECHO 2

 #define OFFSET\_ACI\_CMD\_T\_DTM\_CMD 2

 #define OFFSET\_ACI\_CMD\_T\_SETUP 2

 #define OFFSET\_ACI\_CMD\_T\_WRITE\_DYNAMIC\_DATA 2

 #define OFFSET\_ACI\_CMD\_T\_SET\_LOCAL\_DATA 2

 #define OFFSET\_ACI\_CMD\_T\_CONNECT 2

 #define OFFSET\_ACI\_CMD\_T\_BOND 2

 #define OFFSET\_ACI\_CMD\_T\_DISCONNECT 2

 #define OFFSET\_ACI\_CMD\_T\_SET\_TX\_POWER 2

 #define OFFSET\_ACI\_CMD\_T\_CHANGE\_TIMING 2

 #define OFFSET\_ACI\_CMD\_T\_OPEN\_REMOTE\_PIPE 2

 #define OFFSET\_ACI\_CMD\_T\_SEND\_DATA 2

 #define OFFSET\_ACI\_CMD\_T\_SEND\_DATA\_ACK 2

 #define OFFSET\_ACI\_CMD\_T\_REQUEST\_DATA 2

 #define OFFSET\_ACI\_CMD\_T\_SEND\_DATA\_NACK 2

 #define OFFSET\_ACI\_CMD\_T\_SET\_APP\_LATENCY 2

 #define OFFSET\_ACI\_CMD\_T\_SET\_KEY 2

 #define OFFSET\_ACI\_CMD\_T\_OPEN\_ADV\_PIPE 2

 #define OFFSET\_ACI\_CMD\_T\_BROADCAST 2

 #define OFFSET\_ACI\_CMD\_T\_CLOSE\_REMOTE\_PIPE 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_DEVICE\_STARTED\_T\_DEVICE\_MODE 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_DEVICE\_STARTED\_T\_HW\_ERROR 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_DEVICE\_STARTED\_T\_CREDIT\_AVAILABLE 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_HW\_ERROR\_T\_LINE\_NUM\_LSB 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_HW\_ERROR\_T\_LINE\_NUM\_MSB 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_HW\_ERROR\_T\_FILE\_NAME 2

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_DTM\_CMD\_T\_EVT\_MSB 0

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_DTM\_CMD\_T\_EVT\_LSB 1

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_READ\_DYNAMIC\_DATA\_T\_SEQ\_NO 0

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_READ\_DYNAMIC\_DATA\_T\_DYNAMIC\_DATA 1

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_CONFIGURATION\_ID\_LSB 0

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_CONFIGURATION\_ID\_MSB 1

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_ACI\_VERSION 2

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_FORMAT 3

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_ID\_LSB0 4

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_ID\_LSB1 5

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_ID\_MSB0 6

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_ID\_MSB1 7

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_VERSION\_T\_SETUP\_STATUS 8

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_ADDRESS\_T\_BD\_ADDR\_OWN 0

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_DEVICE\_ADDRESS\_T\_BD\_ADDR\_TYPE 6

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_BATTERY\_LEVEL\_T\_BATTERY\_LEVEL\_LSB 0

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_BATTERY\_LEVEL\_T\_BATTERY\_LEVEL\_MSB 1

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_TEMPERATURE\_T\_TEMPERATURE\_VALUE\_LSB 0

 #define OFFSET\_ACI\_EVT\_CMD\_RSP\_PARAMS\_GET\_TEMPERATURE\_T\_TEMPERATURE\_VALUE\_MSB 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_CMD\_OPCODE 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_CMD\_STATUS 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_DTM\_CMD 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_READ\_DYNAMIC\_DATA 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_VERSION 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_DEVICE\_ADDRESS 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_BATTERY\_LEVEL 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_CMD\_RSP\_T\_GET\_TEMPERATURE 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_DEV\_ADDR\_TYPE 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_DEV\_ADDR 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_RF\_INTERVAL\_LSB 7

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_RF\_INTERVAL\_MSB 8

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_SLAVE\_RF\_LATENCY\_LSB 9

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_SLAVE\_RF\_LATENCY\_MSB 10

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_RF\_TIMEOUT\_LSB 11

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_CONN\_RF\_TIMEOUT\_MSB 12

 #define OFFSET\_ACI\_EVT\_PARAMS\_CONNECTED\_T\_MASTER\_CLOCK\_ACCURACY 13

 #define OFFSET\_ACI\_EVT\_PARAMS\_DISCONNECTED\_T\_ACI\_STATUS 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_DISCONNECTED\_T\_BTLE\_STATUS 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_STATUS\_CODE 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_STATUS\_SOURCE 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_SECMODE1\_BITMAP 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_SECMODE2\_BITMAP 3

 #define OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_KEYS\_EXCHANGED\_SLAVE 4

 #define OFFSET\_ACI\_EVT\_PARAMS\_BOND\_STATUS\_T\_KEYS\_EXCHANGED\_MASTER 5

 #define OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_STATUS\_T\_PIPES\_OPEN\_BITMAP 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_STATUS\_T\_PIPES\_CLOSED\_BITMAP 8

 #define OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_RF\_INTERVAL\_LSB 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_RF\_INTERVAL\_MSB 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_SLAVE\_RF\_LATENCY\_LSB 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_SLAVE\_RF\_LATENCY\_MSB 3

 #define OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_RF\_TIMEOUT\_LSB 4

 #define OFFSET\_ACI\_EVT\_PARAMS\_TIMING\_T\_CONN\_RF\_TIMEOUT\_MSB 5

 #define OFFSET\_ACI\_EVT\_PARAMS\_DATA\_CREDIT\_T\_CREDIT 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_DATA\_ACK\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_DATA\_RECEIVED\_T\_RX\_DATA 0

 #define OFFSET\_ERROR\_DATA\_T\_CONTENT 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_ERROR\_T\_PIPE\_NUMBER 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_ERROR\_T\_ERROR\_CODE 1

 #define OFFSET\_ACI\_EVT\_PARAMS\_PIPE\_ERROR\_T\_ERROR\_DATA 2

 #define OFFSET\_ACI\_EVT\_PARAMS\_DISPLAY\_PASSKEY\_T\_PASSKEY 0

 #define OFFSET\_ACI\_EVT\_PARAMS\_KEY\_REQUEST\_T\_KEY\_TYPE 0

 #define OFFSET\_ACI\_EVT\_T\_LEN 0

 #define OFFSET\_ACI\_EVT\_T\_EVT\_OPCODE 1

 //#define OFFSET\_ACI\_EVT\_T\_EVT\_OPCODE 2

 #define OFFSET\_ACI\_EVT\_T\_DEVICE\_STARTED 2

 #define OFFSET\_ACI\_EVT\_T\_HW\_ERROR 2

 #define OFFSET\_ACI\_EVT\_T\_CMD\_RSP 2

 #define OFFSET\_ACI\_EVT\_T\_CONNECTED 2

 #define OFFSET\_ACI\_EVT\_T\_DISCONNECTED 2

 #define OFFSET\_ACI\_EVT\_T\_BOND\_STATUS 2

 #define OFFSET\_ACI\_EVT\_T\_PIPE\_STATUS 2

 #define OFFSET\_ACI\_EVT\_T\_TIMING 2

 #define OFFSET\_ACI\_EVT\_T\_DATA\_CREDIT 2

 #define OFFSET\_ACI\_EVT\_T\_DATA\_ACK 2

 #define OFFSET\_ACI\_EVT\_T\_DATA\_RECEIVED 2

 #define OFFSET\_ACI\_EVT\_T\_PIPE\_ERROR 2

 #define OFFSET\_ACI\_EVT\_T\_DISPLAY\_PASSKEY 2

 #define OFFSET\_ACI\_EVT\_T\_KEY\_REQUEST 2

#endif //ACI\_OFFSET\_H\_\_

5. aci\_queue.h

#ifndef ACI\_QUEUE\_H\_\_

#define ACI\_QUEUE\_H\_\_

#include "aci.h"

#include "hal\_aci\_tl.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

/\* The ACI\_QUEUE\_SIZE determines the memory usage of the system. \*/

/\* Successfully tested to a ACI\_QUEUE\_SIZE of 4 (interrupt) and 4 (polling) \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*/

#define ACI\_QUEUE\_SIZE 4

/\*\* Data type for queue of data packets to send/receive from radio.

 \*

 \* A FIFO queue is maintained for packets. New packets are added (enqueued)

 \* at the tail and taken (dequeued) from the head. The head variable is the

 \* index of the next packet to dequeue while the tail variable is the index of

 \* where the next packet should be queued.

 \*/

typedef struct {

 hal\_aci\_data\_t aci\_data[ACI\_QUEUE\_SIZE];

 uint8\_t head;

 uint8\_t tail;

} aci\_queue\_t;

void aci\_queue\_init(aci\_queue\_t \*aci\_q);

bool aci\_queue\_dequeue(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data);

bool aci\_queue\_dequeue\_from\_isr(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data);

bool aci\_queue\_enqueue(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data);

bool aci\_queue\_enqueue\_from\_isr(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data);

bool aci\_queue\_is\_empty(aci\_queue\_t \*aci\_q);

bool aci\_queue\_is\_empty\_from\_isr(aci\_queue\_t \*aci\_q);

bool aci\_queue\_is\_full(aci\_queue\_t \*aci\_q);

bool aci\_queue\_is\_full\_from\_isr(aci\_queue\_t \*aci\_q);

bool aci\_queue\_peek(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data);

bool aci\_queue\_peek\_from\_isr(aci\_queue\_t \*aci\_q, hal\_aci\_data\_t \*p\_data);

#endif /\* ACI\_QUEUE\_H\_\_ \*/

/\*\* @} \*/

6. aci\_setup.h

#ifndef H\_ACI\_SETUP

#define H\_ACI\_SETUP

#define SETUP\_SUCCESS 0

#define SETUP\_FAIL\_COMMAND\_QUEUE\_NOT\_EMPTY 1

#define SETUP\_FAIL\_EVENT\_QUEUE\_NOT\_EMPTY 2

#define SETUP\_FAIL\_TIMEOUT 3

#define SETUP\_FAIL\_NOT\_SETUP\_EVENT 4

#define SETUP\_FAIL\_NOT\_COMMAND\_RESPONSE 5

/\*\* @brief Setup the nRF8001 device

 \* @details

 \* Performs ACI Setup by transmitting the setup messages generated by nRFgo Studio to the

 \* nRF8001, and should be called when the nRF8001 starts or resets.

 \* Once all messages are sent, the nRF8001 will send a Device Started Event.

 \* The function requires that the Command queue is empty when it is invoked, and will fail

 \* otherwise.

 \* @returns An integer indicating the reason the function terminated

 \*/

uint8\_t do\_aci\_setup(aci\_state\_t \*aci\_stat);

#endif

7. acilib.h

#ifndef \_acilib\_H\_

#define \_acilib\_H\_

#define MSG\_SET\_LOCAL\_DATA\_BASE\_LEN 2

#define MSG\_CONNECT\_LEN 5

#define MSG\_BOND\_LEN 5

#define MSG\_DISCONNECT\_LEN 2

#define MSG\_BASEBAND\_RESET\_LEN 1

#define MSG\_WAKEUP\_LEN 1

#define MSG\_SET\_RADIO\_TX\_POWER\_LEN 2

#define MSG\_GET\_DEVICE\_ADDR\_LEN 1

#define MSG\_SEND\_DATA\_BASE\_LEN 2

#define MSG\_DATA\_REQUEST\_LEN 2

#define MSG\_OPEN\_REMOTE\_PIPE\_LEN 2

#define MSG\_CLOSE\_REMOTE\_PIPE\_LEN 2

#define MSG\_DTM\_CMD 3

#define MSG\_WRITE\_DYNAMIC\_DATA\_BASE\_LEN 2

#define MSG\_SETUP\_CMD\_BASE\_LEN 1

#define MSG\_ECHO\_MSG\_CMD\_BASE\_LEN 1

#define MSG\_CHANGE\_TIMING\_LEN 9

#define MSG\_SET\_APP\_LATENCY\_LEN 4

#define MSG\_CHANGE\_TIMING\_LEN\_GAP\_PPCP 1

#define MSG\_DIRECT\_CONNECT\_LEN 1

#define MSG\_SET\_KEY\_REJECT\_LEN 2

#define MSG\_SET\_KEY\_PASSKEY\_LEN 8

#define MSG\_SET\_KEY\_OOB\_LEN 18

#define MSG\_ACK\_LEN 2

#define MSG\_NACK\_LEN 3

#define MSG\_BROADCAST\_LEN 5

#define MSG\_OPEN\_ADV\_PIPES\_LEN 9

#endif /\* \_acilib\_H\_ \*/

8. acilib\_defs.h

#ifndef \_acilib\_DEFS\_H\_

#define \_acilib\_DEFS\_H\_

#define ACIL\_DECODE\_EVT\_GET\_LENGTH(buffer\_in) (\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_LEN ))

#define ACIL\_DECODE\_EVT\_GET\_OPCODE(buffer\_in) (\*(buffer\_in + OFFSET\_ACI\_EVT\_T\_EVT\_OPCODE))

#endif /\* \_acilib\_DEFS\_H\_ \*/

9. acilib\_if.h

#ifndef \_acilib\_IF\_H\_

#define \_acilib\_IF\_H\_

/\*\* @brief Encode the ACI message for set test mode command

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] test\_mode Pointer to the test mode in ::aci\_cmd\_params\_test\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_set\_test\_mode(uint8\_t \*buffer, aci\_cmd\_params\_test\_t \*p\_aci\_cmd\_params\_test);

/\*\* @brief Encode the ACI message for sleep command

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_sleep(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message for get device version

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_get\_device\_version(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message for set local data

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_set\_local\_data Pointer to the local data parameters in ::aci\_cmd\_params\_set\_local\_data\_t

 \* @param[in] data\_size Size of data message

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_set\_local\_data(uint8\_t \*buffer, aci\_cmd\_params\_set\_local\_data\_t \*p\_aci\_cmd\_params\_set\_local\_data, uint8\_t data\_size);

/\*\* @brief Encode the ACI message to connect

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_connect Pointer to the run parameters in ::aci\_cmd\_params\_connect\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_connect(uint8\_t \*buffer, aci\_cmd\_params\_connect\_t \*p\_aci\_cmd\_params\_connect);

/\*\* @brief Encode the ACI message to bond

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_bond Pointer to the run parameters in ::aci\_cmd\_params\_bond\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_bond(uint8\_t \*buffer, aci\_cmd\_params\_bond\_t \*p\_aci\_cmd\_params\_bond);

/\*\* @brief Encode the ACI message to disconnect

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_disconnect Pointer to the run parameters in ::aci\_cmd\_params\_disconnect\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_disconnect(uint8\_t \*buffer, aci\_cmd\_params\_disconnect\_t \*p\_aci\_cmd\_params\_disconnect);

/\*\* @brief Encode the ACI message to baseband reset

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

 void acil\_encode\_baseband\_reset(uint8\_t \*buffer);

 /\*\* @brief Encode the ACI message for Directed Advertising

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

 void acil\_encode\_direct\_connect(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message to wakeup

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_wakeup(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message for set radio Tx power

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_set\_tx\_power Pointer to the set Tx power parameters in ::aci\_cmd\_params\_set\_tx\_power\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_set\_radio\_tx\_power(uint8\_t \*buffer, aci\_cmd\_params\_set\_tx\_power\_t \*p\_aci\_cmd\_params\_set\_tx\_power);

/\*\* @brief Encode the ACI message for get device address

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_get\_address(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message for send data

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_send\_data\_t Pointer to the data parameters in ::aci\_cmd\_params\_send\_data\_t

 \* @param[in] data\_size Size of data message

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_send\_data(uint8\_t \*buffer, aci\_cmd\_params\_send\_data\_t \*p\_aci\_cmd\_params\_send\_data\_t, uint8\_t data\_size);

/\*\* @brief Encode the ACI message for request data

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_request\_data Pointer to the request data parameters in ::aci\_cmd\_params\_request\_data\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_request\_data(uint8\_t \*buffer, aci\_cmd\_params\_request\_data\_t \*p\_aci\_cmd\_params\_request\_data);

/\*\* @brief Encode the ACI message for open remote pipe

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_open\_remote\_pipe Pointer to the dynamic data parameters in ::aci\_cmd\_params\_open\_remote\_pipe\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_open\_remote\_pipe(uint8\_t \*buffer, aci\_cmd\_params\_open\_remote\_pipe\_t \*p\_aci\_cmd\_params\_open\_remote\_pipe);

/\*\* @brief Encode the ACI message for close remote pipe

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_close\_remote\_pipe Pointer to the dynamic data parameters in ::aci\_cmd\_params\_close\_remote\_pipe\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_close\_remote\_pipe(uint8\_t \*buffer, aci\_cmd\_params\_close\_remote\_pipe\_t \*p\_aci\_cmd\_params\_close\_remote\_pipe);

/\*\* @brief Encode the ACI message for echo message

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_cmd\_params\_echo Pointer to the dynamic data parameters in ::aci\_cmd\_params\_echo\_t

 \* @param[in] msg\_size Size of the message

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_echo\_msg(uint8\_t \*buffer, aci\_cmd\_params\_echo\_t \*p\_cmd\_params\_echo, uint8\_t msg\_size);

/\*\* @brief Encode the ACI message to battery level

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_battery\_level(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message to temparature

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_temparature(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message to read dynamic data

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_read\_dynamic\_data(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message to change timing request

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_change\_timing Pointer to the change timing parameters in ::aci\_cmd\_params\_change\_timing\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_change\_timing\_req(uint8\_t \*buffer, aci\_cmd\_params\_change\_timing\_t \*p\_aci\_cmd\_params\_change\_timing);

/\*\* @brief Encode the ACI message to change timing request using the timing parameters from GAP PPCP

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_change\_timing Pointer to the change timing parameters in ::aci\_cmd\_params\_change\_timing\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_change\_timing\_req\_GAP\_PPCP(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message for write dynamic data

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] seq\_no Sequence number of the dynamic data (as received in the response to @c Read Dynamic Data)

 \* @param[in] dynamic\_data Pointer to the dynamic data

 \* @param[in] dynamic\_data\_size Size of dynamic data

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_write\_dynamic\_data(uint8\_t \*buffer, uint8\_t seq\_no, uint8\_t\* dynamic\_data, uint8\_t dynamic\_data\_size);

/\*\* @brief Encode the ACI message to send data acknowledgement

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] pipe\_number Pipe number for which the ack is to be sent

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_send\_data\_ack(uint8\_t \*buffer, const uint8\_t pipe\_number);

/\*\* @brief Encode the ACI message to send negative acknowledgement

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] pipe\_number Pipe number for which the nack is to be sent

 \* @param[in] error\_code Error code that has to be sent in the NACK

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_send\_data\_nack(uint8\_t \*buffer, const uint8\_t pipe\_number,const uint8\_t error\_code);

/\*\* @brief Encode the ACI message to set the application latency

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd\_params\_set\_app\_latency Pointer to the set\_application\_latency command parameters in ::aci\_cmd\_params\_dtm\_cmd\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_set\_app\_latency(uint8\_t \*buffer, aci\_cmd\_params\_set\_app\_latency\_t \*p\_aci\_cmd\_params\_set\_app\_latency);

/\*\* @brief Encode the ACI message for setup

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_cmd\_params\_set\_run\_behaviour Pointer to the setup data in ::aci\_cmd\_params\_setup\_t

 \* @param[in] setup\_data\_size Size of setup message

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_setup(uint8\_t \*buffer, aci\_cmd\_params\_setup\_t \*p\_aci\_cmd\_params\_setup, uint8\_t setup\_data\_size);

/\*\* @brief Encode the ACI message for DTM command

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_cmd\_params\_set\_run\_behaviour Pointer to the DTM command parameters in ::aci\_cmd\_params\_dtm\_cmd\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_dtm\_cmd(uint8\_t \*buffer, aci\_cmd\_params\_dtm\_cmd\_t \*p\_aci\_cmd\_params\_dtm\_cmd);

/\*\* @brief Encode the ACI message for Set Key Request command

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_set\_key(uint8\_t \*buffer, aci\_cmd\_params\_set\_key\_t \*p\_aci\_cmd\_params\_set\_key);

/\*\* @brief Encode the ACI message for Bond Security Request command

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_bond\_security\_request(uint8\_t \*buffer);

/\*\* @brief Encode the ACI message

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd Pointer to ACI command data in ::aci\_cmd\_t

 \* @param[in] bool

 \*

 \* @return bool true, if succesful, else returns false

 \*/

bool acil\_encode\_cmd(uint8\_t \*buffer, aci\_cmd\_t \*p\_aci\_cmd);

/\*\* @brief Encode the ACI message for Broadcast command

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd Pointer to ACI command data in ::aci\_cmd\_params\_broadcast\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_broadcast(uint8\_t \*buffer, aci\_cmd\_params\_broadcast\_t \* p\_aci\_cmd\_params\_broadcast);

/\*\* @brief Encode the ACI message for Open Adv Pipes

 \*

 \* @param[in,out] buffer Pointer to ACI message buffer

 \* @param[in] p\_aci\_cmd Pointer to ACI command data in ::aci\_cmd\_params\_open\_adv\_pipe\_t

 \*

 \* @return None

 \*/

void acil\_encode\_cmd\_open\_adv\_pipes(uint8\_t \*buffer, aci\_cmd\_params\_open\_adv\_pipe\_t \* p\_aci\_cmd\_params\_set\_adv\_svc\_data);

/\*\* @brief Decode the ACI event command response

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] buffer Pointer to the decoded message in ::aci\_evt\_params\_cmd\_rsp\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_command\_response(uint8\_t \*buffer\_in, aci\_evt\_params\_cmd\_rsp\_t \*p\_evt\_params\_cmd\_rsp);

/\*\* @brief Decode the ACI event device started

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_aci\_evt Pointer to the decoded message in ::aci\_evt\_params\_device\_started\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_device\_started(uint8\_t \*buffer\_in, aci\_evt\_params\_device\_started\_t \*p\_evt\_params\_device\_started);

/\*\* @brief Decode the ACI event pipe status

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_aci\_evt\_params\_pipe\_status Pointer to the decoded message in ::aci\_evt\_params\_pipe\_status\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_pipe\_status(uint8\_t \*buffer\_in, aci\_evt\_params\_pipe\_status\_t \*p\_aci\_evt\_params\_pipe\_status);

/\*\* @brief Decode the ACI event for disconnected

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_aci\_evt\_params\_disconnected Pointer to the decoded message in ::aci\_evt\_params\_disconnected\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_disconnected(uint8\_t \*buffer\_in, aci\_evt\_params\_disconnected\_t \*p\_aci\_evt\_params\_disconnected);

/\*\* @brief Decode the ACI event for bond status

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_aci\_evt\_params\_bond\_status Pointer to the decoded message in ::aci\_evt\_params\_bond\_status\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_bond\_status(uint8\_t \*buffer\_in, aci\_evt\_params\_bond\_status\_t \*p\_aci\_evt\_params\_bond\_status);

/\*\* @brief Decode the ACI event for data received

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_evt\_params\_data\_received Pointer to the decoded message in ::aci\_evt\_params\_data\_received\_t

 \*

 \* @return size Received data size

 \*/

uint8\_t acil\_decode\_evt\_data\_received(uint8\_t \*buffer\_in, aci\_evt\_params\_data\_received\_t \*p\_evt\_params\_data\_received);

/\*\* @brief Decode the ACI event data acknowledgement

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_evt\_params\_data\_ack Pointer to the decoded message in ::aci\_evt\_params\_data\_ack\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_data\_ack(uint8\_t \*buffer\_in, aci\_evt\_params\_data\_ack\_t \*p\_evt\_params\_data\_ack);

/\*\* @brief Decode the ACI event for hardware error

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_aci\_evt\_params\_hw\_error Pointer to the decoded message in ::aci\_evt\_params\_hw\_error\_t

 \*

 \* @return size Size of debug information

 \*/

uint8\_t acil\_decode\_evt\_hw\_error(uint8\_t \*buffer\_in, aci\_evt\_params\_hw\_error\_t \*p\_aci\_evt\_params\_hw\_error);

/\*\* @brief Decode the ACI event data credit

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_evt\_params\_data\_credit Pointer to the decoded message in ::aci\_evt\_params\_data\_credit\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_credit(uint8\_t \*buffer\_in, aci\_evt\_params\_data\_credit\_t \*p\_evt\_params\_data\_credit);

/\*\* @brief Decode the ACI event for connected

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_aci\_evt\_params\_connected Pointer to the decoded message in ::aci\_evt\_params\_connected\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_connected(uint8\_t \*buffer\_in, aci\_evt\_params\_connected\_t \*p\_aci\_evt\_params\_connected);

/\*\* @brief Decode the ACI event for timing

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_evt\_params\_timing Pointer to the decoded message in ::aci\_evt\_params\_timing\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_timing(uint8\_t \*buffer\_in, aci\_evt\_params\_timing\_t \*p\_evt\_params\_timing);

/\*\* @brief Decode the ACI event for pipe error

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_evt\_params\_pipe\_error Pointer to the decoded message in ::aci\_evt\_params\_pipe\_error\_t

 \*

 \*/

void acil\_decode\_evt\_pipe\_error(uint8\_t \*buffer\_in, aci\_evt\_params\_pipe\_error\_t \*p\_evt\_params\_pipe\_error);

/\*\* @brief Decode the ACI event for key request

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_evt\_params\_key\_type Pointer to the decoded message in ::aci\_evt\_params\_key\_type\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_key\_request(uint8\_t \*buffer\_in, aci\_evt\_params\_key\_request\_t \*p\_evt\_params\_key\_request);

/\*\* @brief Decode the ACI event for echo

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] buffer\_out Pointer to the echo message (max size of buffer ::ACI\_ECHO\_DATA\_MAX\_LEN)

 \*

 \* @return size Received echo message size

 \*/

uint8\_t acil\_decode\_evt\_echo(uint8\_t \*buffer\_in, aci\_evt\_params\_echo\_t \*buffer\_out);

/\*\* @brief Decode the ACI event

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_aci\_evt Pointer to the decoded message in ::aci\_evt\_t

 \*

 \* @return bool true, if succesful, else returns false

 \*/

bool acil\_decode\_evt(uint8\_t \*buffer\_in, aci\_evt\_t \*p\_aci\_evt);

/\*\* @brief Decode the Display Key Event

 \*

 \* @param[in] buffer\_in Pointer to message received

 \* @param[in,out] p\_aci\_evt Pointer to the decoded message in ::aci\_evt\_params\_display\_passkey\_t

 \*

 \* @return None

 \*/

void acil\_decode\_evt\_display\_passkey(uint8\_t \*buffer\_in, aci\_evt\_params\_display\_passkey\_t \*p\_aci\_evt\_params\_display\_passkey);

#endif /\* \_acilib\_IF\_H\_ \*/

10. acilib\_types.h

#ifndef \_acilib\_TYPES\_H\_

#define \_acilib\_TYPES\_H\_

#endif /\* \_acilib\_TYPES\_H\_ \*/

11. ADHEADER.h

#ifndef ADHEADER\_H

#define ADHEADER\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#include <xc.h>

/\*FUNCTION PROTOTYPES\*/

void ADC\_init(void);

unsigned int ADC(void);

/\*FUNCTION DEFINITIONS\*/

void ADC\_init(void){

 AD1PCFG = 0; //set pins as analog

 AD1CON1bits.FORM = 0; //unsigned int

 AD1CON1bits.SSRC = 0; //set to manual conversion

 AD1CON2bits.VCFG = 0; //Use AVdd and AVss

 AD1CON2bits.CSCNA = 0; //Disable Scan Mode

 AD1CON2bits.BUFM = 0; //single 16 bit buffer

 AD1CON2bits.ALTS = 0; //Use mux a

 AD1CON3bits.ADRC = 0; //Use PB Bus Clock

 AD1CON3bits.ADCS = 3; //Math in word doc

 AD1CON1bits.ON = 1;

}

unsigned int ADC(void){

 AD1CHSbits.CH0SA = 4;

 AD1CON1bits.SAMP = 1;

 delay\_us(100);

 AD1CON1bits.SAMP = 0;

 //delay\_us(10);

 while(!AD1CON1bits.DONE);

 return(ADC1BUF0);

}

#ifdef \_\_cplusplus

}

#endif

#endif /\* ADHEADER\_H \*/

12. ble\_assert.h

#ifndef BLE\_ASSERT\_H\_\_

 #define BLE\_ASSERT\_H\_\_

extern void \_\_ble\_assert(const char \*file, uint16\_t line);

#define ble\_assert(expr) \

 ((expr) \

 ? ((void) 0) \

 : \_\_ble\_assert (\_\_FILE\_\_, \_\_LINE\_\_))

#endif /\* BLE\_ASSERT\_H\_\_ \*/

13. bleconfigbitsrev2014vC.h

#ifndef CONFIGBITS\_H

#define CONFIGBITS\_H

/\*

 \* REv 8 boards.

 \* resonator is 8 MHz

 \* Will switch to internal if external not present or fails

 \* internal (FRC) clock

 peripher clock = at 10 MHz (80 MHz/8)

 \*/

#pragma config FNOSC = FRCPLL // Oscillator selection

#pragma config POSCMOD = OFF // Primary oscillator mode

#pragma config FPLLIDIV = DIV\_2 // PLL input divider (8 -> 4)

#pragma config FPLLMUL = MUL\_20 // PLL multiplier ( 4x20 = 80)

#pragma config FPLLODIV = DIV\_1 // PLL output divider (80)

#pragma config FPBDIV = DIV\_4 // Peripheral bus clock divider 20 mhz (80/4)

#pragma config FSOSCEN = OFF // Secondary oscillator enable

/\* Clock control settings

\*/

#pragma config IESO = ON // Internal/external clock switchover

#pragma config FCKSM = CSECME // Clock switching (CSx)/Clock monitor (CMx)

#pragma config OSCIOFNC = OFF // Clock output on OSCO pin enable

/\* USB Settings

\*/

#pragma config UPLLEN = OFF // USB PLL enable

#pragma config UPLLIDIV = DIV\_2 // USB PLL input divider

#pragma config FVBUSONIO = OFF // VBUS pin control

#pragma config FUSBIDIO = OFF // USBID pin control

// Other Peripheral Device settings

#pragma config FWDTEN = OFF // Watchdog timer enable

//#pragma config WDTPS = PS4096 // Watchdog timer post-scaler

#pragma config WDTPS = PS2048 // Watchdog timer post-scaler

#pragma config FSRSSEL = PRIORITY\_7 // SRS interrupt priority

#pragma config DEBUG = ON

 //There was a multiple declaration problem between these declarations and sleepmodeHEADER.h

#pragma config ICESEL = ICS\_PGx1 // ICE pin selection

#endif /\* CONFIGBITS\_H \*/

14. hal\_aci\_tl.h

#ifndef HAL\_ACI\_TL\_H\_\_

#define HAL\_ACI\_TL\_H\_\_

//#include "hal\_platform.h"

#include "aci.h"

//#include "boards.h"

#ifndef HAL\_ACI\_MAX\_LENGTH

#define HAL\_ACI\_MAX\_LENGTH 31

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Unused nRF8001 pin \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define UNUSED 255

/\*\* Data type for ACI commands and events \*/

typedef struct

{

 uint8\_t status\_byte;

 uint8\_t buffer[HAL\_ACI\_MAX\_LENGTH+1];

} \_aci\_packed\_ hal\_aci\_data\_t;

ACI\_ASSERT\_SIZE(hal\_aci\_data\_t, HAL\_ACI\_MAX\_LENGTH + 2);

/\*\* Datatype for ACI pins and interface (polling/interrupt)\*/

typedef struct aci\_pins\_t

{

 uint8\_t board\_name; //Optional : Use BOARD\_DEFAULT if you do not know. See boards.h

 uint8\_t reqn\_pin; //Required

 uint8\_t rdyn\_pin; //Required

 uint8\_t mosi\_pin; //Required

 uint8\_t miso\_pin; //Required

 uint8\_t sck\_pin; //Required

 uint8\_t spi\_clock\_divider; //Required : Clock divider on the SPI clock : nRF8001 supports a maximum clock of 3MHz

 uint8\_t reset\_pin; //Recommended but optional - Set it to UNUSED when not connected

 uint8\_t active\_pin; //Optional - Set it to UNUSED when not connected

 uint8\_t optional\_chip\_sel\_pin; //Optional - Used only when the reqn line is required to be separate from the SPI chip select. Eg. Arduino DUE

 bool interface\_is\_interrupt; //Required - true = Uses interrupt on RDYN pin. false - Uses polling on RDYN pin

 uint8\_t interrupt\_number; //Required when using interrupts, otherwise ignored

} aci\_pins\_t ;

/\*\* @brief ACI Transport Layer initialization.

 \* @details

 \* This function initializes the transport layer, including configuring the SPI, creating

 \* message queues for Commands and Events and setting up interrupt if required.

 \* @param a\_pins Pins on the MCU used to connect to the nRF8001

 \* @param bool True if debug printing should be enabled on the Serial.

 \*/

void hal\_aci\_tl\_init(aci\_pins\_t \*a\_pins, bool debug);

/\*\* @brief Sends an ACI command to the radio.

 \* @details

 \* This function sends an ACI command to the radio. This queue up the message to send and

 \* lower the request line. When the device lowers the ready line, @ref m\_aci\_spi\_transfer()

 \* will send the data.

 \* @param aci\_buffer Pointer to the message to send.

 \* @return True if the data was successfully queued for sending,

 \* false if there is no more space to store messages to send.

 \*/

bool hal\_aci\_tl\_send(hal\_aci\_data\_t \*aci\_buffer);

/\*\* @brief Process pending transactions.

 \* @details

 \* The library code takes care of calling this function to check if the nRF8001 RDYN line indicates a

 \* pending transaction. It will send a pending message if there is one and return any receive message

 \* that was pending.

 \* @return Points to data buffer for received data. Length byte in buffer is 0 if no data received.

 \*/

hal\_aci\_data\_t \* hal\_aci\_tl\_poll\_get(void);

/\*\* @brief Get an ACI event from the event queue

 \* @details

 \* Call this function from the main context to get an event from the ACI event queue

 \* This is called by lib\_aci\_event\_get

 \*/

bool hal\_aci\_tl\_event\_get(hal\_aci\_data\_t \*p\_aci\_data);

/\*\* @brief Peek an ACI event from the event queue

 \* @details

 \* Call this function from the main context to peek an event from the ACI event queue.

 \* This is called by lib\_aci\_event\_peek

 \*/

bool hal\_aci\_tl\_event\_peek(hal\_aci\_data\_t \*p\_aci\_data);

/\*\* @brief Enable debug printing of all ACI commands sent and ACI events received

 \* @details

 \* when the enable parameter is true. The debug printing is enabled on the Serial.

 \* When the enable parameter is false. The debug printing is disabled on the Serial.

 \* By default the debug printing is disabled.

 \*/

void hal\_aci\_tl\_debug\_print(bool enable);

/\*\* @brief Pin reset the nRF8001

 \* @details

 \* The reset line of the nF8001 needs to kept low for 200 ns.

 \* Redbearlab shield v1.1 and v2012.07 are exceptions as they

 \* have a Power ON Reset circuit that works differently.

 \* The function handles the exceptions based on the board\_name in aci\_pins\_t

 \*/

void hal\_aci\_tl\_pin\_reset(void);

/\*\* @brief Return full status of transmit queue

 \* @details

 \*

 \*/

 bool hal\_aci\_tl\_rx\_q\_full(void);

 /\*\* @brief Return empty status of receive queue

 \* @details

 \*

 \*/

 bool hal\_aci\_tl\_rx\_q\_empty(void);

/\*\* @brief Return full status of receive queue

 \* @details

 \*

 \*/

 bool hal\_aci\_tl\_tx\_q\_full(void);

 /\*\* @brief Return empty status of transmit queue

 \* @details

 \*

 \*/

 bool hal\_aci\_tl\_tx\_q\_empty(void);

/\*\* @brief Flush the ACI command Queue and the ACI Event Queue

 \* @details

 \* Call this function in the main thread

 \*/

void hal\_aci\_tl\_q\_flush(void);

#endif // HAL\_ACI\_TL\_H\_\_

/\*\* @} \*/

15. hal\_platform.h

#define PLATFORM\_H\_\_

//Board dependent defines

 #include <stdint.h>

 #include <stdbool.h>

 #include <string.h>

 // #include <wiring.h>

 //#include <WProgram.h>

 //For making the Serial.Print compatible between Arduino and Chipkit

 #define F(X) (X)

 //For ChipKit neither PROGMEM or PSTR are needed for PIC32

 #define PROGMEM

 #define PSTR(s) (s)

 #define pgm\_read\_byte(x) (\*((char \*)x))

 #define pgm\_read\_byte\_near(x) (\*((char \*)x))

 #define pgm\_read\_byte\_far(x) (\*((char \*)x))

 #define pgm\_read\_word(x) (\*((short \*)x))

 #define pgm\_read\_word\_near(x) (\*((short \*)x))

 #define pgm\_read\_workd\_far(x) (\*((short \*)x))

 #define prog\_void const void

 #define prog\_char const char

 #define prog\_uchar const unsigned char

 #define prog\_int8\_t const int8\_t

 #define prog\_uint8\_t const uint8\_t

 #define prog\_int16\_t const int16\_t

 #define prog\_uint16\_t const uint16\_t

 #define prog\_int32\_t const int32\_t

 #define prog\_uint32\_t const uint32\_t

 #define prog\_int64\_t const int64\_t

 #define prog\_uint64\_t const uint64\_t

 //Redefine the function for reading from flash in ChipKit

 #define memcpy\_P memcpy

//#endif /\* PLATFORM\_H\_\_ \*/

16. lib\_aci.h

#ifndef LIB\_ACI\_H\_\_

#define LIB\_ACI\_H\_\_

#include <stdbool.h>

//#define true 1

//#define false 0

//#define bool \_Bool // need this since converting from C++

/\*\* @file

\* @brief ACI library

\*/

/\*\* @addtogroup lib\_aci

@{

@brief Library for the logical part of the Application Controller Interface (ACI)

\*/

#include "hal\_platform.h"

#include "hal\_aci\_tl.h"

#include "aci\_queue.h"

#include "aci.h"

#include "aci\_cmds.h"

#include "aci\_evts.h"

#define EVT\_CMD\_RESPONSE\_MIN\_LENGTH 3

#define PIPES\_ARRAY\_SIZE ((ACI\_DEVICE\_MAX\_PIPES + 7)/8)

/\* Same size as a hal\_aci\_data\_t \*/

typedef struct {

 uint8\_t debug\_byte;

 aci\_evt\_t evt;

} \_aci\_packed\_ hal\_aci\_evt\_t;

ACI\_ASSERT\_SIZE(hal\_aci\_evt\_t, 34);

typedef struct

{

 uint8\_t location; /\*\*< enum aci\_pipe\_store\_t \*/

 aci\_pipe\_type\_t pipe\_type;

} services\_pipe\_type\_mapping\_t;

typedef struct aci\_setup\_info\_t

{

 services\_pipe\_type\_mapping\_t \*services\_pipe\_type\_mapping;

 uint8\_t number\_of\_pipes;

 hal\_aci\_data\_t \*setup\_msgs;

 uint8\_t num\_setup\_msgs;

} aci\_setup\_info\_t;

// aci\_struct that will contain

// total initial credits

// current credit

// current state of the aci (setup/standby/active/sleep)

// open remote pipe pending

// close remote pipe pending

// Current pipe available bitmap

// Current pipe closed bitmap

// Current connection interval, slave latency and link supervision timeout

// Current State of the the GATT client (Service Discovery status)

// Relationship of bond to peer address

typedef struct aci\_state\_t

{

 aci\_pins\_t aci\_pins; /\* Pins on the MCU used to connect to the nRF8001 \*/

 aci\_setup\_info\_t aci\_setup\_info; /\* Data structures that are created from nRFgo Studio \*/

 uint8\_t bonded; /\* ( aci\_bond\_status\_code\_t ) Is the nRF8001 bonded to a peer device \*/

 uint8\_t data\_credit\_total; /\* Total data credit available for the specific version of the nRF8001, total equals available when a link is established \*/

 aci\_device\_operation\_mode\_t device\_state; /\* Operating mode of the nRF8001 \*/

 /\* \*/

 /\* Start : Variables that are valid only when in a connection \*/

 uint8\_t data\_credit\_available; /\* Available data credits at a specific point of time, ACI\_EVT\_DATA\_CREDIT updates the available credits \*/

 uint16\_t connection\_interval; /\* Multiply by 1.25 to get the connection interval in milliseconds\*/

 uint16\_t slave\_latency; /\* Number of consecutive connection intervals that the nRF8001 is not required to transmit. Use this to save power \*/

 uint16\_t supervision\_timeout; /\* Multiply by 10 to get the supervision timeout in milliseconds \*/

 uint8\_t pipes\_open\_bitmap[PIPES\_ARRAY\_SIZE]; /\* Bitmap -> pipes are open and can be used for sending data over the air \*/

 uint8\_t pipes\_closed\_bitmap[PIPES\_ARRAY\_SIZE]; /\* Bitmap -> pipes are closed and cannot be used for sending data over the air \*/

 bool confirmation\_pending; /\* Attribute protocol Handle Value confirmation is pending for a Handle Value Indication

 (ACK is pending for a TX\_ACK pipe) on local GATT Server\*/

 /\* End : Variables that are valid only when in a connection \*/

} aci\_state\_t;

#define DISCONNECT\_REASON\_CX\_TIMEOUT 0x08

#define DISCONNECT\_REASON\_CX\_CLOSED\_BY\_PEER\_DEVICE 0x13

#define DISCONNECT\_REASON\_POWER\_LOSS 0x14

#define DISCONNECT\_REASON\_CX\_CLOSED\_BY\_LOCAL\_DEVICE 0x16

#define DISCONNECT\_REASON\_ADVERTISER\_TIMEOUT 0x50

/\*\* @name Functions for library management \*/

//@{

/\*\* @brief Function to enable printing of all ACI commands sent and ACI events received

 \* @details This function shall be used to enable or disable the debug printing.

 Debug printing is disabled by default.

 \*/

void lib\_aci\_debug\_print(bool enable);

/\*\* @brief Function to pin reset the nRF8001

 \* @details Pin resets the nRF8001 also handles differences between development boards

 \*/

void lib\_aci\_pin\_reset(void);

/\*\* @brief Initialization function.

 \* @details This function shall be used to initialize/reset ACI Library and also Resets the

 \* nRF8001 by togging the reset pin of the nRF8001. This function will reset

 \* all the variables locally used by ACI library to their respective default values.

 \* @param bool True if the data was successfully queued for sending,

 \* false if there is no more space to store messages to send.

 \*/

void lib\_aci\_init(aci\_state\_t \*aci\_stat, bool debug);

/\*\* @brief Gets the number of currently available ACI credits.

 \* @return Number of ACI credits.

 \*/

uint8\_t lib\_aci\_get\_nb\_available\_credits(aci\_state\_t \*aci\_stat);

/\*\* @brief Gets the connection interval in milliseconds.

 \* @return Connection interval in milliseconds.

 \*/

uint16\_t lib\_aci\_get\_cx\_interval\_ms(aci\_state\_t \*aci\_stat);

/\*\* @brief Gets the connection interval in multiple of 1.25&nbsp;ms.

 \* @return Connection interval in multiple of 1.25&nbsp;ms.

 \*/

uint16\_t lib\_aci\_get\_cx\_interval(aci\_state\_t \*aci\_stat);

/\*\* @brief Gets the current slave latency.

 \* @return Current slave latency.

 \*/

uint16\_t lib\_aci\_get\_slave\_latency(aci\_state\_t \*aci\_stat);

/\*\* @brief Checks if a given pipe is available.

 \* @param pipe Pipe to check.

 \* @return True if the pipe is available, otherwise false.

 \*/

bool lib\_aci\_is\_pipe\_available(aci\_state\_t \*aci\_stat, uint8\_t pipe);

/\*\* @brief Checks if a given pipe is closed.

 \* @param pipe Pipe to check.

 \* @return True if the pipe is closed, otherwise false.

 \*/

bool lib\_aci\_is\_pipe\_closed(aci\_state\_t \*aci\_stat, uint8\_t pipe);

/\*\* @brief Checks if the discovery operation is finished.

 \* @return True if the discovery is finished.

 \*/

bool lib\_aci\_is\_discovery\_finished(aci\_state\_t \*aci\_stat);

//@}

/\*\* @name ACI Commands available in all modes \*/

//@{

/\*\* @brief Sets the radio in sleep mode.

 \* @details The function sends a @c sleep command to the radio.

 \* If the radio is advertising or connected, it sends back an error, then use lib\_aci\_radio\_reset

 \* if advertising or disconnect if in a connection.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_sleep(void);

/\*\* @brief Resets the radio.

 \* @details The function sends a @c BasebandReset command to the radio.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_radio\_reset(void);

/\*\* @brief Radio starts directed advertising to bonded device.

 \* @details The function sends a @c DirectedConnect command to the radio.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_direct\_connect(void);

/\*\* @brief Gets the radio's version.

 \* @details This function sends a @c GetDeviceVersion command.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_device\_version(void);

/\*\* @brief Gets the device address.

 \* @details This function sends a @c GetDeviceAddress command.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_get\_address(void);

/\*\* @brief Gets the temperature.

 \* @details This function sends a @c GetTemperature command. lib\_aci

 \* calls the @ref lib\_aci\_transaction\_finished\_hook() function when the temperature is received.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_get\_temperature(void);

/\*\* @brief Gets the battery level.

 \* @details This function sends a @c GetBatteryLevel command.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_get\_battery\_level(void);

//@}

/\*\* @name ACI commands available in Sleep mode \*/

//@{

/\*\* @brief Wakes up the radio.

 \* @details This function sends a @c Wakeup command to wake up the radio from

 \* sleep mode. When woken up the radio sends a @c DeviceStartedEvent and

 \* a @c CommandResponseEvent.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_wakeup(void);

//@}

/\*\* @name ACI commands available in Active mode \*/

//@{

/\*\* @brief Sets the radio in test mode.

 \* @details This function sends a @c Test command to the radio. There are two

 \* Test modes available:

 \* - UART: DTM commands are received over UART.

 \* - ACI: DTM commands are received over ACI.

 \* The same command is used to exit the test mode When receiving

 \* a @c DeviceStartedEvent the radio has entered the new mode.

 \* @param enter\_exit\_test\_mode Enter a Test mode, or exit Test mode.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_test(aci\_test\_mode\_change\_t enter\_exit\_test\_mode);

/\*\* @brief Sets the radio's TX power.

 \* @details This function sends a @c SetTxPower command.

 \* @param tx\_power TX power to be used by the radio.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_set\_tx\_power(aci\_device\_output\_power\_t tx\_power);

/\*\* @brief Tries to connect to a peer device.

 \* @details This function sends a @c Connect command to the radio.

 \* @param run\_timeout Maximum advertising time in seconds (0 means infinite).

 \* @param adv\_interval Advertising interval (in multiple of 0.625&nbsp;ms).

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_connect(uint16\_t run\_timeout, uint16\_t adv\_interval);

/\*\* @brief Tries to bond with a peer device.

 \* @details This function sends a @c Bond command to the radio.

 \* @param run\_timeout Maximum advertising time in seconds (0 means infinite).

 \* @param adv\_interval Advertising interval (in multiple of 0.625&nbsp;ms).

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_bond(uint16\_t run\_timeout, uint16\_t adv\_interval);

/\*\* @brief Disconnects from peer device.

 \* @details This function sends a @c Disconnect command to the radio.

 \* @param reason Reason for disconnecting.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_disconnect(aci\_state\_t \*aci\_stat, aci\_disconnect\_reason\_t reason);

/\*\*@brief Sets Local Data.

 \* @details

 \* This command updates the value of the characteristic value or the characteristic descriptor stored locally on the device.

 \* Can be called for all types of pipes as long as the data is stored locally.

 \* @param ACI state structure

 \* @param pipe Pipe number on which the data should be set.

 \* @param value Pointer to the data to set.

 \* @param size Size of the data to set.

 \* @return True if the transaction is successfully initiated.

\*/

bool lib\_aci\_set\_local\_data(aci\_state\_t \*aci\_stat, uint8\_t pipe, uint8\_t \*value, uint8\_t size);

/\*\* @brief Sends Broadcast message to the radio.

 \* @details The Broadcast message starts advertisement procedure

 \* using the given interval with the intention of broadcasting data to a peer device.

 \* @param timeout Time, in seconds, to advertise before exiting to standby mode (0 means infinite).

 \* Valid values: 0 to 16383.

 \* @param adv\_interval Advertising interval (in multiple of 0.625&nbsp;ms).

 \* Valid values: 160 to 16384 (which corresponds to an interval from 100 ms to 10.24 s).

 \* @return True if the broadcast message is sent successfully to the radio.

\*/

bool lib\_aci\_broadcast(const uint16\_t timeout, const uint16\_t adv\_interval);

/\*\* @name Open Advertising Pipes. \*/

/\*\* @brief Sends a command to the radio to set the input pipe to be placed in Advertisement Service Data.

 \* @details This function sends a command to the radio that places the pipe in

 \* advertisement service data. To start advertising service data, call this function before

 \* Connecting, Broadcasting or Bonding to peer. The data that should be sent in the advertisement packets

 \* must be set using the @c lib\_aci\_set\_local\_data function. This function can be called during

 \* advertising to enable/disable broadcast pipes.

 \* @param pipe The pipe that has to be placed in advertising service data.

 \* @return True if the Open Adv Pipe message is sent successfully to the radio.

\*/

bool lib\_aci\_open\_adv\_pipe(const uint8\_t pipe);

/\*\* @name Open Advertising Pipes \*/

/\*\* @brief Sends a command to the radio to set the pipes to be placed in Advertisement Service Data.

 \* @details This function will send a command to the radio that will set the pipes to be placed in

 \* advertisement Service Data. To start advertising service data, this function should be called before

 \* Connecting, Broadcasting or Bonding to peer. This function can be called during

 \* advertising to enable/disable broadcast pipes. Use this as an alternative to @ref lib\_aci\_open\_adv\_pipe

 \* to avoid multiple function calls for placing multiple pipes in the adv data.

 \* @param adv\_service\_data\_pipes Pipe bitmap, where '1' indicates that the corresponding

 \* Valid Values: 0000000000000000 to FEFFFFFFFFFFFF7F (See the ACI Pipe Status Evt bitmap in the nRF8001 datasheet

 \* TX\_BROADCAST pipe data is to be placed in Advertising Service Data fields

 \* @return true if the Open Adv Pipe message was sent successfully to the radio.

\*/

bool lib\_aci\_open\_adv\_pipes(const uint8\_t \* const adv\_service\_data\_pipes);

//@}

/\*\* @name ACI commands available in Connected mode \*/

//@{

/\*\* @brief Sets a given application latency.

 \* @details This function sends a @c setApplicationLatency command.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_set\_app\_latency(uint16\_t latency, aci\_app\_latency\_mode\_t latency\_mode);

/\*\* @brief Opens a remote pipe.

 \* @details This function sends an @c OpenRemotePipe command.

 \* @param pipe Number of the pipe to open.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_open\_remote\_pipe(aci\_state\_t \*aci\_stat, uint8\_t pipe);

/\*\* @brief Closes a remote pipe.

 \* @details This function sends an @c CloseRemotePipe command.

 \* @param pipe Pipe number to close.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_close\_remote\_pipe(aci\_state\_t \*aci\_stat, uint8\_t pipe);

/\*\* @brief Sends data on a given pipe.

 \* @details This function sends a @c SendData command with application data to

 \* the radio. This function memorizes credit use, and checks that

 \* enough credits are available.

 \* @param pipe Pipe number on which the data should be sent.

 \* @param value Pointer to the data to send.

 \* @param size Size of the data to send.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_send\_data(uint8\_t pipe, uint8\_t \*value, uint8\_t size);

/\*\* @brief Requests data from a given pipe.

 \* @details This function sends a @c RequestData command to the radio. This

 \* function memorizes credit uses, and check that enough credits are available.

 \* After this command, the radio sends back either a @c DataReceivedEvent

 \* or a @c PipeErrorEvent.

 \* @param pipe Pipe number on which the data is requested.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_request\_data(aci\_state\_t \*aci\_stat, uint8\_t pipe);

/\*\* @brief Sends a L2CAP change connection parameters request.

 \* @details This function sends a @c ChangeTiming command to the radio. This command triggers a "L2CAP change connection parameters" request

 \* to the master. If the master rejects or accepts but doesn't change the connection parameters within

 \* 30 seconds, a timing event with the unchanged connection parameters is sent by the radio.

 \* If the request is accepted and the master changes connection parameters, a timing event with

 \* the new connection parameters is sent by the radio.

 \* If the master doesn't reply to the request within 60 seconds, the radio disconnects.

 \* @param minimun\_cx\_interval Minimum connection interval requested, in multiple of 1.25&nbsp;ms.

 \* @param maximum\_cx\_interval Maximum connection interval requested, in multiple of 1.25&nbsp;ms.

 \* @param slave\_latency requested slave latency.

 \* @param timeout requested slave timeout, in multiple of 10&nbsp;ms.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_change\_timing(uint16\_t minimun\_cx\_interval, uint16\_t maximum\_cx\_interval, uint16\_t slave\_latency, uint16\_t timeout);

/\*\* @brief Sends a L2CAP change connection parameters request with the connection predefined preffered connection parameters.

 \* @details This function sends a @c ChangeTiming command to the radio. This command triggers a "L2CAP change connection parameters" request

 \* to the master. If the master rejects or accepts but doesn't change the connection parameters within

 \* 30 seconds, a timing event with the unchanged connection parameters is sent by the radio.

 \* If the request is accepted and the master changes connection parameters, a timing event with

 \* the new connection parameters is sent by the radio.

 \* If the master doesn't reply to the request within 60 seconds, the radio disconnects.

 \* The timing parameters used are the Timing parameters in the GAP settings in the nRFgo Studio.

 \* The Timing parameters as stored as the GAP Preferred Peripheral Connection Parameters.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_change\_timing\_GAP\_PPCP(void);

/\*\* @brief Sends acknowledgement message to peer.

 \* @details This function sends @c SendDataAck command to radio. The radio is expected

 \* to send either Handle Value Confirmation or Write response depending

 \* on whether the data is stored remotely or locally.

 \* @param pipe Pipe number for which the acknowledgement is to be sent.

 \* @return True if the ack was sent successfully. False otherwise.

\*/

bool lib\_aci\_send\_ack(aci\_state\_t \*aci\_stat, const uint8\_t pipe);

/\*\* @brief Sends negative acknowledgement message to peer.

 \* @details This function sends @c SendDataNack command to radio. The radio is expected

 \* to send Error Response to the peer.

 \* @param pipe Pipe number for which the nack is to be sent.

 \* @param error\_code Error code to be sent in the NACk.

 \* @return True if the nack was sent successfully. False otherwise.

\*/

bool lib\_aci\_send\_nack(aci\_state\_t \*aci\_stat, const uint8\_t pipe, const uint8\_t error\_code);

/\*\* @brief Sends ReadDynamicData command to the host.

 \* @details This function sends @c ReadDynamicData command to host. The host is expected

 \* to send @c CommandResponse back with the dynamic data. The application is expected to

 \* call this function in a loop until all the dynamic data is read out from the host.

 \* As long as there is dynamic data to be read from the host, the command response

 \* for this message has its status field set to ACI\_STATUS\_TRANSACTION\_CONTINUE (0x01).

 \* The application may chose to store this read out data in a non-volatile memory location

 \* and later chose to write it back using the function lib\_aci\_write\_dynamic\_data.

 \* @return True if the command was sent successfully through the ACI. False otherwise.

\*/

bool lib\_aci\_read\_dynamic\_data(void);

/\*\* @brief Sends WriteDynamicData command to the host.

 \* @details This function sends @c WriteDynamicData command to host. The host is expected

 \* to send @c CommandResponse with the status of this operation. As long as the status field

 \* in the @c CommandResponse is ACI\_STATUS\_TRANSACTION\_CONTINUE (0x01), the hosts expects

 \* more dynamic data to be written. This function should ideally be called in a cycle,

 \* until all the stored dynamic data is sent to the host. This function should be

 \* called with the dynamic data obtained from the response to a @c ReadDynamicData

 \* (see @c lib\_aci\_read\_dynamic\_data) command.

 \* @param sequence\_number Sequence number of the dynamic data to be sent.

 \* @param dynamic\_data Pointer to the dynamic data.

 \* @param length Length of the dynamic data.

 \* @return True if the command was sent successfully through the ACI. False otherwise.

\*/

bool lib\_aci\_write\_dynamic\_data(uint8\_t sequence\_number, uint8\_t\* dynamic\_data, uint8\_t length);

//@}

/\*\* @name ACI commands available while connected in Bond mode \*/

//@{

/\*\* @brief Sends a SMP Security Request.

 \* @details This function send a @c BondRequest command to the radio.

 \* This command triggers a SMP Security Request to the master. If the

 \* master rejects with a pairing failed or if the bond timer expires the connection is closed.

 \* @return True if the transaction is successfully initiated.

 \*/

bool lib\_aci\_bond\_request(void);

/\*\* @brief Set the key requested by the 8001.

 \* @details This function sends an @c SetKey command to the radio.

 \* @param key\_rsp\_type Type of key.

 \* @param key Pointer to the key to set.

 \* @param len Length of the key.

 \* @return True if the transaction is successfully initiated.

\*/

bool lib\_aci\_set\_key(aci\_key\_type\_t key\_rsp\_type, uint8\_t \*key, uint8\_t len);

//@}

/\*\* @name ACI commands available in Test mode \*/

//@{

/\*\* @brief Sends an echo message

 \* @details This function sends an @c Echo command to the radio. lib\_aci

 \* places the Echp ACI command in the ACI command queue

 \* @param message\_size Length of the data to send.

 \* @param message\_data Pointer to the data to send.

 \* @return True if the transaction is successfully initiated.

\*/

bool lib\_aci\_echo\_msg(uint8\_t message\_size, uint8\_t \*message\_data);

/\*\* @brief Sends an DTM command

 \* @details This function sends an @c DTM command to the radio.

 \* @param dtm\_command\_msbyte Most significant byte of the DTM command.

 \* @param dtm\_command\_lsbyte Least significant byte of the DTM command.

 \* @return True if the transaction is successfully initiated.

\*/

bool lib\_aci\_dtm\_command(uint8\_t dtm\_command\_msbyte, uint8\_t dtm\_command\_lsbyte);

/\*\* @brief Gets an ACI event from the ACI Event Queue

 \* @details This function gets an ACI event from the ACI event queue.

 \* The queue is updated by the SPI driver for the ACI running in the interrupt context

 \* @param aci\_stat pointer to the state of the ACI.

 \* @param p\_aci\_data pointer to the ACI Event. The ACI Event received will be copied into this pointer.

 \* @return True if an ACI Event was copied to the pointer.

\*/

bool lib\_aci\_event\_get(aci\_state\_t \*aci\_stat, hal\_aci\_evt\_t \* aci\_evt);

/\*\* @brief Peeks an ACI event from the ACI Event Queue

 \* @details This function peeks at the top event in the ACI event queue.

 \* In polling mode, this function will query the nRF8001 for pending events, but unlike

 \* lib\_aci\_event\_get() it will not dequeue the event from the local queue, but will instead

 \* only peek at it.

 \* @return True if an ACI Event was copied to the pointer.

\*/

bool lib\_aci\_event\_peek(hal\_aci\_evt\_t \*p\_aci\_evt\_data);

/\*\* @brief Flushes the events in the ACI command queues and ACI Event queue

 \*

\*/

void lib\_aci\_flush(void);

/\*\* @brief Return full status of the Event queue

 \* @details

 \*

 \*/

 bool lib\_aci\_event\_queue\_full(void);

 /\*\* @brief Return empty status of the Event queue

 \* @details

 \*

 \*/

 bool lib\_aci\_event\_queue\_empty(void);

/\*\* @brief Return full status of Command queue

 \* @details

 \*

 \*/

 bool lib\_aci\_command\_queue\_full(void);

 /\*\* @brief Return empty status of Command queue

 \* @details

 \*

 \*/

 bool lib\_aci\_command\_queue\_empty(void);

//@}

/\*\* @} \*/

#endif /\* LIB\_ACI\_H\_\_ \*/

17. newtooth2.h

#include "bleconfigbitsrev2014vC.h"

#include "SDlib.h"

#include "SDlib\_delay.h"

#include "lib\_aci.h"

#include "aci\_setup.h"

#include "uart\_over\_ble.h"

#include <sys/attribs.h>

#include <xc.h>

#include <stdbool.h>

//#define true 1

//#define false 0

//#define bool \_Bool // need this since converting from C++

/\*\*

Put the nRF8001 setup in the RAM of the nRF8001.

\*/

#include "services.h"

/\*\*

Include the services\_lock.h to put the setup in the OTP memory of the nRF8001.

This would mean that the setup cannot be changed once put in.

However this removes the need to do the setup of the nRF8001 on every reset.

\*/

void serial\_init6(int br);

unsigned long available(void);

uint8\_t read(void);

void serialEvent(void);

void bluetoothINIT(void);

int bluetoothCONNECT(void);

int bluetoothSEND(void);

void loop(float);

void setup(void);

// bluetoothSEND(int byte);

#ifdef SERVICES\_PIPE\_TYPE\_MAPPING\_CONTENT

 static services\_pipe\_type\_mapping\_t

 services\_pipe\_type\_mapping[NUMBER\_OF\_PIPES] = SERVICES\_PIPE\_TYPE\_MAPPING\_CONTENT;

#else

 #define NUMBER\_OF\_PIPES 0

 static services\_pipe\_type\_mapping\_t \* services\_pipe\_type\_mapping = NULL;

#endif

/\* Store the setup for the nRF8001 in the flash of the AVR to save on RAM \*/

static hal\_aci\_data\_t setup\_msgs[NB\_SETUP\_MESSAGES] PROGMEM = SETUP\_MESSAGES\_CONTENT;

// aci\_struct that will contain

// total initial credits

// current credit

// current state of the aci (setup/standby/active/sleep)

// open remote pipe pending

// close remote pipe pending

// Current pipe available bitmap

// Current pipe closed bitmap

// Current connection interval, slave latency and link supervision timeout

// Current State of the the GATT client (Service Discovery)

// Status of the bond (R) Peer address

static struct aci\_state\_t aci\_state;

/\*

Temporary buffers for sending ACI commands

\*/

static hal\_aci\_evt\_t aci\_data;

//static hal\_aci\_data\_t aci\_cmd;

/\*

Timing change state variable

\*/

static bool timing\_change\_done = false;

/\*

Used to test the UART TX characteristic notification

\*/

static uart\_over\_ble\_t uart\_over\_ble;

static uint8\_t uart\_buffer[20];

static uint8\_t uart\_buffer\_len = 0;

static uint8\_t dummychar = 0;

/\*

Initialize the radio\_ack. This is the ack received for every transmitted packet.

\*/

//static bool radio\_ack\_pending = false;

/\* Define how assert should function in the BLE library \*/

void \_\_ble\_assert(const char \*file, uint16\_t line)

{

 //Serial.print("ERROR ");

 printf("ERROR ");

 //Serial.print(file);

 printf("%c", file);

 //Serial.print(": ");

 printf(": ");

 // Serial.print(line);

 printf("%s", line);

 //Serial.print("\n");

 printf("\n");

 while(1);

}

/\*

Description:

In this template we are using the BTLE as a UART and can send and receive packets.

The maximum size of a packet is 20 bytes.

When a command it received a response(s) are transmitted back.

Since the response is done using a Notification the peer must have opened it(subscribed to it) before any packet is transmitted.

The pipe for the UART\_TX becomes available once the peer opens it.

See section 20.4.1 -> Opening a Transmit pipe

In the master control panel, clicking Enable Services will open all the pipes on the nRF8001.

The ACI Evt Data Credit provides the radio level ack of a transmitted packet.

\*/

void serial\_init6(int br)

{

/\* setting bits for the U6MODE register \*/

U6MODE=0; //Clear all the bits in the register to 0.

U6MODEbits.ON=1; //Enable the UART

U6MODEbits.BRGH =1; //Set the UART to high speed mode

/\*Setting bits for the U6STA register\*/

U6STA=0; //Clear all the bits in the register to 0.

U6STAbits.URXEN=1; //Enable the Receive

U6STAbits.UTXEN =1; // Enable the Transmit

/\*Get the Peripheral Bus Clock Frequency\*/

unsigned long fpb;

fpb=get\_pb\_clock();

/\*Set up the baud rate register\*/

U6BRG=(fpb/(4\*br))-1; //no floor function because it will truncate automatically

}

void setup(void)

{

 //Serial.begin(115200);

 serial\_init6(57600);// correct baudrate for 2.5 Mhz and fpb of 20Mhz changed from 115200 to 57600

 //Wait until the serial port is available (useful only for the Leonardo)

 //As the Leonardo board is not reseted every time you open the Serial Monitor

 #if defined (\_\_AVR\_ATmega32U4\_\_)

 while(!Serial)

 {}

 delay\_ms(5000); //5 seconds delay for enabling to see the start up comments on the serial board

 #elif defined(\_\_PIC32MX\_\_)

 delay\_ms(1000);

 #endif

 //Serial.println(F("Arduino setup"));

 //Serial.println(F("Set line ending to newline to send data from the serial monitor"));

 /\*\*

 Point ACI data structures to the the setup data that the nRFgo studio generated for the nRF8001

 \*/

 if (NULL != services\_pipe\_type\_mapping)

 {

 aci\_state.aci\_setup\_info.services\_pipe\_type\_mapping = &services\_pipe\_type\_mapping[0];

 }

 else

 {

 aci\_state.aci\_setup\_info.services\_pipe\_type\_mapping = NULL;

 }

 aci\_state.aci\_setup\_info.number\_of\_pipes = NUMBER\_OF\_PIPES;

 aci\_state.aci\_setup\_info.setup\_msgs = setup\_msgs;

 aci\_state.aci\_setup\_info.num\_setup\_msgs = NB\_SETUP\_MESSAGES;

 /\*

 Tell the ACI library, the MCU to nRF8001 pin connections.

 The Active pin is optional and can be marked UNUSED

 \*/

 aci\_state.aci\_pins.board\_name = 0; //See board.h for details REDBEARLAB\_SHIELD\_V1\_1 or BOARD\_DEFAULT

 aci\_state.aci\_pins.reqn\_pin = LATBbits.LATB8; // REQN (LATBbits.B8 or PORTBbits.RB8????

 // LATBbits.LATB8=1;

 aci\_state.aci\_pins.rdyn\_pin = PORTDbits.RD0; // RDYN (input to microcontroller, from nRF8001)

 //aci\_state.aci\_pins.mosi\_pin = LATFbits.LATF5; // use LAT to write to the MOSI pin

 aci\_state.aci\_pins.mosi\_pin = SPI4BUF; // use LAT to write to the MOSI pin

 //aci\_state.aci\_pins.mosi\_pin = PORTFbits.F4; // use PORT to read from the MISO pin

 aci\_state.aci\_pins.miso\_pin = SPI4BUF;

 //aci\_state.aci\_pins.sck\_pin = LATBbits.LATB14; // according to kitboard description

 // aci\_state.aci\_pins.spi\_clock\_divider = SPI\_CLOCK\_DIV8;//Left as SPI\_CLOCK\_DIV8 (2.5MHz)

 //SPI\_CLOCK\_DIV16 = 1MHz SPI speed

 aci\_state.aci\_pins.reset\_pin = LATBbits.LATB12; //4 for Nordic board, UNUSED for REDBEARLAB\_SHIELD\_V1\_1

 aci\_state.aci\_pins.active\_pin = LATBbits.LATB11;

 aci\_state.aci\_pins.optional\_chip\_sel\_pin = UNUSED;

 aci\_state.aci\_pins.interface\_is\_interrupt = false; //Interrupts still not available in Chipkit

 aci\_state.aci\_pins.interrupt\_number = 1;

 //We reset the nRF8001 here by toggling the RESET line connected to the nRF8001

 //If the RESET line is not available we call the ACI Radio Reset to soft reset the nRF8001

 //then we initialize the data structures required to setup the nRF8001

 //The second parameter is for turning debug printing on for the ACI Commands and Events so they be printed on the Serial

 lib\_aci\_init(&aci\_state, false);

 //Serial.println(F("Set up done"));

}

void uart\_over\_ble\_init(void)

{

 uart\_over\_ble.uart\_rts\_local = true;

}

// a place to start looking for bluetooth send

bool uart\_tx(uint8\_t \*buffer, uint8\_t buffer\_len)

{

 bool status = false;

 if (lib\_aci\_is\_pipe\_available(&aci\_state, PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX) &&

 (aci\_state.data\_credit\_available >= 1))

 {

 status = lib\_aci\_send\_data(PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX, buffer, buffer\_len);

 if (status)

 {

 aci\_state.data\_credit\_available--;

 }

 }

 return status;

}

bool uart\_process\_control\_point\_rx(uint8\_t \*byte, uint8\_t length)

{

 bool status = false;

 aci\_ll\_conn\_params\_t \*conn\_params;

 if (lib\_aci\_is\_pipe\_available(&aci\_state, PIPE\_UART\_OVER\_BTLE\_UART\_CONTROL\_POINT\_TX) )

 {

 //Serial.println(\*byte, HEX);

 //printf(" %p \n ",( void \* )byte );// How to make this hex...?; is this necessary

 switch(\*byte)

 {

 /\*

 Queues a ACI Disconnect to the nRF8001 when this packet is received.

 May cause some of the UART packets being sent to be dropped

 \*/

 case UART\_OVER\_BLE\_DISCONNECT:

 /\*

 Parameters:

 None

 \*/

 lib\_aci\_disconnect(&aci\_state, ACI\_REASON\_TERMINATE);

 status = true;

 break;

 /\*

 Queues an ACI Change Timing to the nRF8001

 \*/

 case UART\_OVER\_BLE\_LINK\_TIMING\_REQ:

 /\*

 Parameters:

 Connection interval min: 2 bytes

 Connection interval max: 2 bytes

 Slave latency: 2 bytes

 Timeout: 2 bytes

 Same format as Peripheral Preferred Connection Parameters (See nRFgo studio -> nRF8001 Configuration -> GAP Settings

 Refer to the ACI Change Timing Request in the nRF8001 Product Specifications

 \*/

 conn\_params = (aci\_ll\_conn\_params\_t \*)(byte+1);

 lib\_aci\_change\_timing( conn\_params->min\_conn\_interval,

 conn\_params->max\_conn\_interval,

 conn\_params->slave\_latency,

 conn\_params->timeout\_mult);

 status = true;

 break;

 /\*

 Clears the RTS of the UART over BLE

 \*/

 case UART\_OVER\_BLE\_TRANSMIT\_STOP:

 /\*

 Parameters:

 None

 \*/

 uart\_over\_ble.uart\_rts\_local = false;

 status = true;

 break;

 /\*

 Set the RTS of the UART over BLE

 \*/

 case UART\_OVER\_BLE\_TRANSMIT\_OK:

 /\*

 Parameters:

 None

 \*/

 uart\_over\_ble.uart\_rts\_local = true;

 status = true;

 break;

 }

 }

 return status;

}

struct aci\_return {

 bool sent;

 int count;

 bool connected;

};

struct aci\_return aci\_loop(struct aci\_return results, float data)

{

 static bool setup\_required = false;

 //printf("did this work?"); YES! but don't do again eleanor its an infinite loop

 // We enter the if statement only when there is a ACI event available to be processed

 if (lib\_aci\_event\_get(&aci\_state, &aci\_data))

 {

 aci\_evt\_t \* aci\_evt;

 aci\_evt = &aci\_data.evt;

 //aci\_evt->evt\_opcode = 0x81;

 //printf(" 0x%02x ", aci\_evt->evt\_opcode); //4-0-3

 switch(aci\_evt->evt\_opcode)

 {

 /\*\*

 As soon as you reset the nRF8001 you will get an ACI Device Started Event

 \*/

 case ACI\_EVT\_DEVICE\_STARTED:

 {

 //printf("Case1 \n");

 aci\_state.data\_credit\_total = aci\_evt->params.device\_started.credit\_available;

 switch(aci\_evt->params.device\_started.device\_mode)

 {

 case ACI\_DEVICE\_SETUP:

 /\*\*

 When the device is in the setup mode

 \*/

 //Serial.println(F("Evt Device Started: Setup"));

 printf("Evt Device Started: Setup \n");

 setup\_required = true;

 break;

 case ACI\_DEVICE\_STANDBY:

 // Serial.println(F("Evt Device Started: Standby"));

 printf("Evt Device Started: Standby\n");

 //Looking for an iPhone by sending radio advertisements

 //When an iPhone connects to us we will get an ACI\_EVT\_CONNECTED event from the nRF8001

 if (aci\_evt->params.device\_started.hw\_error)

 {

 delay\_ms(20); //Handle the HW error event correctly.

 }

 else

 {

 lib\_aci\_connect(0/\* in seconds : 0 means forever \*/, 0x0050 /\* advertising interval 50ms\*/);

 //Serial.println(F("Advertising started : Tap Connect on the nRF UART app"));

 printf("Advertising started : Tap Connect on the nRF UART app\n");

 }

 break;

 }

 }

 break; //ACI Device Started Event

 case ACI\_EVT\_CMD\_RSP:

 //printf("Case2");

 //If an ACI command response event comes with an error -> stop

 if (ACI\_STATUS\_SUCCESS != aci\_evt->params.cmd\_rsp.cmd\_status)

 {

 //ACI ReadDynamicData and ACI WriteDynamicData will have status codes of

 //TRANSACTION\_CONTINUE and TRANSACTION\_COMPLETE

 //all other ACI commands will have status code of ACI\_STATUS\_SCUCCESS for a successful command

 //Serial.print(F("ACI Command "));

 printf("ACI Command \n");

 //Serial.println(aci\_evt->params.cmd\_rsp.cmd\_opcode, HEX);

 //printf(""); I have no idea what to do about this one

 //Serial.print(F("Evt Cmd respone: Status "));

 printf("Evt Cmd respone: Status \n");

 //Serial.println(aci\_evt->params.cmd\_rsp.cmd\_status, HEX);

 printf(" %p ", (void \*)aci\_evt->params.cmd\_rsp.cmd\_status);

 }

 //printf(" %p \n",aci\_evt->params.cmd\_rsp.cmd\_opcode );

 if (ACI\_CMD\_GET\_DEVICE\_VERSION == aci\_evt->params.cmd\_rsp.cmd\_opcode)

 {

 // printf("Does get to this if\n"); // does not come into this if 11:09 12/11/14

 //Store the version and configuration information of the nRF8001 in the Hardware Revision String Characteristic

 lib\_aci\_set\_local\_data(&aci\_state, PIPE\_DEVICE\_INFORMATION\_HARDWARE\_REVISION\_STRING\_SET,

 (uint8\_t \*)&(aci\_evt->params.cmd\_rsp.params.get\_device\_version), sizeof(aci\_evt\_cmd\_rsp\_params\_get\_device\_version\_t));

 }

 break;

 case ACI\_EVT\_CONNECTED:

 //Serial.println(F("Evt Connected"));

 printf("Evt Connected\n");

 results.sent = false;

 results.connected = true;

 uart\_over\_ble\_init();

 timing\_change\_done = false;

 aci\_state.data\_credit\_available = aci\_state.data\_credit\_total;

 /\*

 Get the device version of the nRF8001 and store it in the Hardware Revision String

 \*/

 lib\_aci\_device\_version();

 break;

 case ACI\_EVT\_PIPE\_STATUS:

 //Serial.println(F("Evt Pipe Status"));

 results.count += results.count;

 printf("Evt Pipe Status\n");

 if (lib\_aci\_is\_pipe\_available(&aci\_state, PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX) && (false == timing\_change\_done))

 {

 lib\_aci\_change\_timing\_GAP\_PPCP(); // change the timing on the link as specified in the nRFgo studio -> nRF8001 conf. -> GAP.

 // Used to increase or decrease bandwidth

 timing\_change\_done = true;

 //char hello[]="6,7,3,5,7,8,9\n";

 char hello[4];

 sprintf(hello, "%f", data);

 uart\_tx((uint8\_t \*)&hello[0], strlen(hello));

 //Serial.print(F("Sending :"));

 printf("Sending :\n");

 //Serial.println(hello);

 printf( "%s\n", hello);

 }

 break;

 case ACI\_EVT\_TIMING:

 //Serial.println(F("Evt link connection interval changed"));

 printf("Evt link connection interval changed\n");

 lib\_aci\_set\_local\_data(&aci\_state,

 PIPE\_UART\_OVER\_BTLE\_UART\_LINK\_TIMING\_CURRENT\_SET,

 (uint8\_t \*)&(aci\_evt->params.timing.conn\_rf\_interval), /\* Byte aligned \*/

 PIPE\_UART\_OVER\_BTLE\_UART\_LINK\_TIMING\_CURRENT\_SET\_MAX\_SIZE);

 break;

 case ACI\_EVT\_DISCONNECTED:

 //Serial.println(F("Evt Disconnected/Advertising timed out"));

 printf("Evt Disconnected/Advertising timed out\n");

 lib\_aci\_connect(0/\* in seconds : 0 means forever \*/, 0x0050 /\* advertising interval 50ms\*/);

 //Serial.println(F("Advertising started. Tap Connect on the nRF UART app"));

 printf("Advertising started. Tap Connect on the nRF UART app\n");

 break;

 case ACI\_EVT\_DATA\_RECEIVED:

 // Serial.print(F("Pipe Number: "));

 //printf("Pipe Number: ");

 //Serial.println(aci\_evt->params.data\_received.rx\_data.pipe\_number, DEC);

 //printf(" %p \n ",aci\_evt->params.data\_received.rx\_data.pipe\_number); //how to print these pointers correctly

// Commented out (void \*) before aci\_evt in line above: CMS 12/8 2:18pm, also lin 518 an 522 in this file, 537 542

 if (PIPE\_UART\_OVER\_BTLE\_UART\_RX\_RX == aci\_evt->params.data\_received.rx\_data.pipe\_number)

 {

 printf("data received");

 //Serial.print(F(" Data(Hex) : "));

 printf(" Data (from phone): ");

 int i ;

 for(i; (i< aci\_evt->len - 2); i++)

 {

 //Serial.print((char)aci\_evt->params.data\_received.rx\_data.aci\_data[i]);

 printf("%c",( char \* )aci\_evt->params.data\_received.rx\_data.aci\_data[i] );

 uart\_buffer[i] = aci\_evt->params.data\_received.rx\_data.aci\_data[i];

 //Serial.print(F(" "));

 //printf(" ");

 }

 uart\_buffer\_len = aci\_evt->len - 2;

 //Serial.println(F(""));

 printf("\n");

 if (lib\_aci\_is\_pipe\_available(&aci\_state, PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX))

 {

 /\*Do this to test the loopback otherwise comment it out\*/

 /\*

 if (!uart\_tx(&uart\_buffer[0], aci\_evt->len - 2))

 {

 Serial.println(F("UART loopback failed"));

 }

 else

 {

 Serial.println(F("UART loopback OK"));

 }

 \*/

 }

 }

 if (PIPE\_UART\_OVER\_BTLE\_UART\_CONTROL\_POINT\_RX == aci\_evt->params.data\_received.rx\_data.pipe\_number)

 {

 uart\_process\_control\_point\_rx(&aci\_evt->params.data\_received.rx\_data.aci\_data[0], aci\_evt->len - 2); //Subtract for Opcode and Pipe number

 }

 break;

 case ACI\_EVT\_DATA\_CREDIT:

 //printf("Case456789");

 aci\_state.data\_credit\_available = aci\_state.data\_credit\_available + aci\_evt->params.data\_credit.credit;

 break;

 case ACI\_EVT\_PIPE\_ERROR:

 //See the appendix in the nRF8001 Product Specication for details on the error codes

 //Serial.print(F("ACI Evt Pipe Error: Pipe #:"));

 printf("ACI Evt Pipe Error: Pipe #:");

 // Serial.print(aci\_evt->params.pipe\_error.pipe\_number, DEC);

 // printf(" %p ", aci\_evt->params.pipe\_error.pipe\_number);

 //Serial.print(F(" Pipe Error Code: 0x"));

 //printf(" Pipe Error Code: 0x");

 //Serial.println(aci\_evt->params.pipe\_error.error\_code, HEX);

 printf(" %08x ", aci\_evt->params.pipe\_error.error\_code);

 //Increment the credit available as the data packet was not sent.

 //The pipe error also represents the Attribute protocol Error Response sent from the peer and that should not be counted

 //for the credit.

 if (ACI\_STATUS\_ERROR\_PEER\_ATT\_ERROR != aci\_evt->params.pipe\_error.error\_code)

 {

 aci\_state.data\_credit\_available++;

 }

 break;

 case ACI\_EVT\_HW\_ERROR:

 //Serial.print(F("HW error: "));

 printf("HW error: ");

 //Serial.println(aci\_evt->params.hw\_error.line\_num, DEC);

 printf("%p", aci\_evt->params.hw\_error.line\_num);

 uint8\_t counter = 0;

 for(counter; counter <= (aci\_evt->len - 3); counter++)

 {

 //Serial.write(aci\_evt->params.hw\_error.file\_name[counter]); //uint8\_t file\_name[20];

 printf(" %s ",aci\_evt->params.hw\_error.file\_name[counter]);

 }

 //Serial.println();

 lib\_aci\_connect(0/\* in seconds, 0 means forever \*/, 0x0050 /\* advertising interval 50ms\*/);

 //Serial.println(F("Advertising started. Tap Connect on the nRF UART app"));

 printf("Advertising started. Tap Connect on the nRF UART app\n");

 break;

 }

 }

 else

 {

 char data\_tx[10];

 snprintf(data\_tx, sizeof(data\_tx), "%f", data);

// char data = I2C\_read();

 if ((lib\_aci\_is\_pipe\_available(&aci\_state, PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX)) && results.count < 10)

 {

 uart\_tx((uint8\_t\*)&data\_tx, strlen(data\_tx));

 printf("transfer");

 results.count = results.count + 1;

 printf("%d",results.count);

 }

 //Serial.println(F("No ACI Events available"));

 // No event in the ACI Event queue and if there is no event in the ACI command queue the arduino can go to sleep

 // Arduino can go to sleep now

 // Wakeup from sleep from the RDYN line

 /\*

 if (lib\_aci\_is\_pipe\_available(&aci\_state, PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX))

 {

 timing\_change\_done = true;

 //char hello[]="6,7,3,5,7,8,9\n";

 char hello[4];

 sprintf(hello, "%f", data);

 uart\_tx((uint8\_t \*)&hello[0], strlen(hello));

 //Serial.print(F("Sending :"));

 printf("Sending :\n");

 //Serial.println(hello);

 printf( "%s\n", hello);

 } \*/

 }

 /\* setup\_required is set to true when the device starts up and enters setup mode.

 \* It indicates that do\_aci\_setup() should be called. The flag should be cleared if

 \* do\_aci\_setup() returns ACI\_STATUS\_TRANSACTION\_COMPLETE.

 \*/

 if(setup\_required)

 {

 if (SETUP\_SUCCESS == do\_aci\_setup(&aci\_state))

 {

 setup\_required = false;

 }

 }

 if(results.count > 1)

 results.sent = true;

 return results;

}

bool stringComplete = false; // whether the string is complete

uint8\_t stringIndex = 0; //Initialize the index to store incoming chars

/\*

void loop(float data) {

 //Process any ACI commands or events

 aci\_loop(data);

 // print the string when a newline arrives:

 if (stringComplete)

 {

 //Serial.print(F("Sending: "));

 printf("Sending: ");

 //Serial.println((char \*)&uart\_buffer[0]);

 printf(" %p ", (char \*)&uart\_buffer[0]);

 uart\_buffer\_len = stringIndex + 1;

 if (!lib\_aci\_send\_data(PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX, uart\_buffer, uart\_buffer\_len))

 {

 //Serial.println(F("Serial input dropped"));

 printf("Serial input dropped");

 }

 // clear the uart\_buffer:

 for (stringIndex = 0; stringIndex < 20; stringIndex++)

 {

 uart\_buffer[stringIndex] = ' ';

 }

 // reset the flag and the index in order to receive more data

 stringIndex = 0;

 stringComplete = false;

 }

 //For ChipKit you have to call the function that reads from Serial

 #if defined (\_\_PIC32MX\_\_)

 if (available())

 {

 serialEvent();

 }

 #endif

} \*/

/\*

 COMMENT ONLY FOR ARDUINO

 SerialEvent occurs whenever a new data comes in the

 hardware serial RX. This routine is run between each

 time loop() runs, so using delay inside loop can delay

 response. Multiple bytes of data may be available.

 Serial Event is NOT compatible with Leonardo, Micro, Esplora

 \*/

void serialEvent(void) {

 while(available() > 0){

 // get the new byte:

 uint8\_t dummychar=read();

 if(!stringComplete)

 {

 if (dummychar == '\n')

 {

 // if the incoming character is a newline, set a flag

 // so the main loop can do something about it

 stringIndex--;

 stringComplete = true;

 }

 else

 {

 if(stringIndex > 19)

 {

 //Serial.println("Serial input truncated");

 printf("Serial input truncated");

 stringIndex--;

 stringComplete = true;

 }

 else

 {

 // add it to the uart\_buffer

 uart\_buffer[stringIndex] = dummychar;

 stringIndex++;

 }

 }

 }

 }

}

bool ble\_connect(float data) {

 struct aci\_return results;

 results.sent = false;

 results.count = 0;

 results.connected = false;

 while(results.sent == false)

 {

 //printf("false...\r\n");

 results = aci\_loop(results, data);

 if(results.connected == true) {

 printf("connected \n");

 break;

 }

 }

 return results.connected;

}

bool ble\_transfer(float data) {

 struct aci\_return results;

 results.sent = false;

 results.count = 0;

 //results.connected = false;

 while(results.sent == false)

 {

 //printf("false...\r\n");

 results = aci\_loop(results, data);

 //printf("results.sent: %d\n\r",results.sent);

 if(results.sent == true)

 printf("sent test 1 \n");

 }

 return results.sent;

}

/\*

void main()

{

 bool trash;

 float data = 5.395;

 setup();

 trash = ble\_connect(data);

 trash = ble\_transfer(data);

}\*/

 /\*

 int CONNECT = 0;

 int SENT = 0;

 setup();

 while(1){

 CONNECT = bluetoothCONNECT();

 if(CONNECT)

 break;

 }

 while(1){

 CONNECT = bluetoothCONNECT();

 SENT = loop2();

 if(SENT)

 break;

 }

 while(1){

 printf('great success');

 }

}

 \* \*/

unsigned long available(void)

{

 unsigned long dataAvailable = 0;

 if (U6STAbits.URXDA)

 dataAvailable = 1;

 else

 dataAvailable =0;

 return dataAvailable;

}

uint8\_t read(void)

{

 unsigned int data= U6RXREG;

 return data;

}

18. SDlib\_delay.h

#ifndef \_DELAY\_H\_

#define \_DELAY\_H\_

#include <xc.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Delay functions for kit32 board \*/

/\* delay routines\*/

void set\_sys\_clock(unsigned long val);

unsigned long get\_sys\_clock(void);

void set\_pb\_clock(unsigned long val);

unsigned long get\_pb\_clock(void);

void delay\_ms(unsigned long val);

void delay\_us(unsigned long val);

#endif //DELAY\_H

19. services.h

#ifndef SETUP\_MESSAGES\_H\_\_

#define SETUP\_MESSAGES\_H\_\_

#include "hal\_platform.h"

#include "aci.h"

#define SETUP\_ID 0

#define SETUP\_FORMAT 2 /\*\* nRF8001 Cx \*/

#define ACI\_DYNAMIC\_DATA\_SIZE 263

/\* Service: GATT - Characteristic: Service Changed - Pipe: TX\_ACK \*/

#define PIPE\_GATT\_SERVICE\_CHANGED\_TX\_ACK 1

#define PIPE\_GATT\_SERVICE\_CHANGED\_TX\_ACK\_MAX\_SIZE 4

/\* Service: Device Information - Characteristic: Hardware Revision String - Pipe: SET \*/

#define PIPE\_DEVICE\_INFORMATION\_HARDWARE\_REVISION\_STRING\_SET 2

#define PIPE\_DEVICE\_INFORMATION\_HARDWARE\_REVISION\_STRING\_SET\_MAX\_SIZE 9

/\* Service: Device Information - Characteristic: Manufacturer Name String - Pipe: SET \*/

#define PIPE\_DEVICE\_INFORMATION\_MANUFACTURER\_NAME\_STRING\_SET 3

#define PIPE\_DEVICE\_INFORMATION\_MANUFACTURER\_NAME\_STRING\_SET\_MAX\_SIZE 20

/\* Service: Device Information - Characteristic: Model Number String - Pipe: SET \*/

#define PIPE\_DEVICE\_INFORMATION\_MODEL\_NUMBER\_STRING\_SET 4

#define PIPE\_DEVICE\_INFORMATION\_MODEL\_NUMBER\_STRING\_SET\_MAX\_SIZE 8

/\* Service: Device Information - Characteristic: Firmware Revision String - Pipe: SET \*/

#define PIPE\_DEVICE\_INFORMATION\_FIRMWARE\_REVISION\_STRING\_SET 5

#define PIPE\_DEVICE\_INFORMATION\_FIRMWARE\_REVISION\_STRING\_SET\_MAX\_SIZE 4

/\* Service: Device Information - Characteristic: PNP\_ID - Pipe: SET \*/

#define PIPE\_DEVICE\_INFORMATION\_PNP\_ID\_SET 6

#define PIPE\_DEVICE\_INFORMATION\_PNP\_ID\_SET\_MAX\_SIZE 7

/\* Service: UART over BTLE - Characteristic: UART RX - Pipe: RX \*/

#define PIPE\_UART\_OVER\_BTLE\_UART\_RX\_RX 7

#define PIPE\_UART\_OVER\_BTLE\_UART\_RX\_RX\_MAX\_SIZE 20

/\* Service: UART over BTLE - Characteristic: UART TX - Pipe: TX \*/

#define PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX 8

#define PIPE\_UART\_OVER\_BTLE\_UART\_TX\_TX\_MAX\_SIZE 20

/\* Service: UART over BTLE - Characteristic: UART Control Point - Pipe: TX \*/

#define PIPE\_UART\_OVER\_BTLE\_UART\_CONTROL\_POINT\_TX 9

#define PIPE\_UART\_OVER\_BTLE\_UART\_CONTROL\_POINT\_TX\_MAX\_SIZE 9

/\* Service: UART over BTLE - Characteristic: UART Control Point - Pipe: RX \*/

#define PIPE\_UART\_OVER\_BTLE\_UART\_CONTROL\_POINT\_RX 10

#define PIPE\_UART\_OVER\_BTLE\_UART\_CONTROL\_POINT\_RX\_MAX\_SIZE 9

/\* Service: UART over BTLE - Characteristic: UART Link Timing Current - Pipe: SET \*/

#define PIPE\_UART\_OVER\_BTLE\_UART\_LINK\_TIMING\_CURRENT\_SET 11

#define PIPE\_UART\_OVER\_BTLE\_UART\_LINK\_TIMING\_CURRENT\_SET\_MAX\_SIZE 6

#define NUMBER\_OF\_PIPES 11

#define SERVICES\_PIPE\_TYPE\_MAPPING\_CONTENT {\

 {ACI\_STORE\_LOCAL, ACI\_TX\_ACK}, \

 {ACI\_STORE\_LOCAL, ACI\_SET}, \

 {ACI\_STORE\_LOCAL, ACI\_SET}, \

 {ACI\_STORE\_LOCAL, ACI\_SET}, \

 {ACI\_STORE\_LOCAL, ACI\_SET}, \

 {ACI\_STORE\_LOCAL, ACI\_SET}, \

 {ACI\_STORE\_LOCAL, ACI\_RX}, \

 {ACI\_STORE\_LOCAL, ACI\_TX}, \

 {ACI\_STORE\_LOCAL, ACI\_TX}, \

 {ACI\_STORE\_LOCAL, ACI\_RX}, \

 {ACI\_STORE\_LOCAL, ACI\_SET}, \

}

#define GAP\_PPCP\_MAX\_CONN\_INT 0x12 /\*\*< Maximum connection interval as a multiple of 1.25 msec , 0xFFFF means no specific value requested \*/

#define GAP\_PPCP\_MIN\_CONN\_INT 0xa /\*\*< Minimum connection interval as a multiple of 1.25 msec , 0xFFFF means no specific value requested \*/

#define GAP\_PPCP\_SLAVE\_LATENCY 0

#define GAP\_PPCP\_CONN\_TIMEOUT 0xa /\*\* Connection Supervision timeout multiplier as a multiple of 10msec, 0xFFFF means no specific value requested \*/

#define NB\_SETUP\_MESSAGES 30

#define SETUP\_MESSAGES\_CONTENT {\

 {0x00,\

 {\

 0x07,0x06,0x00,0x00,0x02,0x02,0x41,0xfe,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0a,0x00,0x0b,0x01,0x01,0x00,0x00,0x06,0x00,0x00,\

 0x90,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x1e,0x06,0x10,0x1c,0x01,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,\

 0x00,0x00,0x00,0x10,0x00,0x00,0x00,0x14,0x03,0x90,0x01,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0x00,0x04,0x04,0x02,0x02,0x00,0x01,0x28,0x00,0x01,0x00,0x18,0x04,0x04,0x05,0x05,0x00,\

 0x02,0x28,0x03,0x01,0x02,0x03,0x00,0x00,0x2a,0x04,0x04,0x14,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0x1c,0x05,0x00,0x03,0x2a,0x00,0x01,0x48,0x65,0x6c,0x6c,0x6f,0x63,0x73,0x65,0x6d,0x69,\

 0x2e,0x63,0x6f,0x6d,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x04,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0x38,0x05,0x05,0x00,0x04,0x28,0x03,0x01,0x02,0x05,0x00,0x01,0x2a,0x06,0x04,0x03,0x02,\

 0x00,0x05,0x2a,0x01,0x01,0x80,0x00,0x04,0x04,0x05,0x05,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0x54,0x06,0x28,0x03,0x01,0x02,0x07,0x00,0x04,0x2a,0x06,0x04,0x09,0x08,0x00,0x07,0x2a,\

 0x04,0x01,0x0a,0x00,0x12,0x00,0x00,0x00,0x0a,0x00,0x04,0x04,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0x70,0x02,0x02,0x00,0x08,0x28,0x00,0x01,0x01,0x18,0x04,0x04,0x05,0x05,0x00,0x09,0x28,\

 0x03,0x01,0x22,0x0a,0x00,0x05,0x2a,0x26,0x04,0x05,0x04,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0x8c,0x0a,0x2a,0x05,0x01,0x00,0x00,0x00,0x00,0x46,0x14,0x03,0x02,0x00,0x0b,0x29,0x02,\

 0x01,0x00,0x00,0x04,0x04,0x02,0x02,0x00,0x0c,0x28,0x00,0x01,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0xa8,0x0a,0x18,0x04,0x04,0x05,0x05,0x00,0x0d,0x28,0x03,0x01,0x02,0x0e,0x00,0x27,0x2a,\

 0x04,0x04,0x09,0x01,0x00,0x0e,0x2a,0x27,0x01,0x0a,0x00,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0xc4,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x04,0x05,0x05,0x00,0x0f,0x28,0x03,0x01,0x02,\

 0x10,0x00,0x29,0x2a,0x04,0x04,0x14,0x02,0x00,0x10,0x2a,0x29,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0xe0,0x01,0x30,0x31,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,\

 0x00,0x00,0x00,0x00,0x00,0x04,0x04,0x05,0x05,0x00,0x11,0x28,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x20,0xfc,0x03,0x01,0x02,0x12,0x00,0x24,0x2a,0x04,0x04,0x08,0x02,0x00,0x12,0x2a,0x24,0x01,\

 0x31,0x32,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x04,0x05,0x05,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0x18,0x00,0x13,0x28,0x03,0x01,0x02,0x14,0x00,0x26,0x2a,0x04,0x04,0x04,0x02,0x00,0x14,\

 0x2a,0x26,0x01,0x33,0x34,0x00,0x00,0x04,0x04,0x05,0x05,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0x34,0x15,0x28,0x03,0x01,0x02,0x16,0x00,0x50,0x2a,0x06,0x04,0x08,0x07,0x00,0x16,0x2a,\

 0x50,0x01,0x02,0x00,0x00,0xaa,0xaa,0xcc,0xcc,0x04,0x04,0x10,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0x50,0x10,0x00,0x17,0x28,0x00,0x01,0x9e,0xca,0xdc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,0xf3,\

 0xa3,0xb5,0x01,0x00,0x40,0x6e,0x04,0x04,0x13,0x13,0x00,0x18,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0x6c,0x28,0x03,0x01,0x04,0x19,0x00,0x9e,0xca,0xdc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,0xf3,\

 0xa3,0xb5,0x02,0x00,0x40,0x6e,0x44,0x10,0x14,0x00,0x00,0x19,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0x88,0x00,0x02,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,\

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x04,0x04,0x13,0x13,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0xa4,0x1a,0x28,0x03,0x01,0x10,0x1b,0x00,0x9e,0xca,0xdc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,\

 0xf3,0xa3,0xb5,0x03,0x00,0x40,0x6e,0x14,0x00,0x14,0x00,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0xc0,0x1b,0x00,0x03,0x02,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,\

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x14,0x03,0x02,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0xdc,0x00,0x1c,0x29,0x02,0x01,0x00,0x00,0x04,0x04,0x13,0x13,0x00,0x1d,0x28,0x03,0x01,\

 0x14,0x1e,0x00,0x9e,0xca,0xdc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x21,0xf8,0xf3,0xa3,0xb5,0x04,0x00,0x40,0x6e,0x54,0x10,0x09,0x00,0x00,0x1e,0x00,0x04,0x02,\

 0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x46,0x14,0x03,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x22,0x14,0x02,0x00,0x1f,0x29,0x02,0x01,0x00,0x00,0x04,0x04,0x13,0x13,0x00,0x20,0x28,0x03,\

 0x01,0x02,0x21,0x00,0x9e,0xca,0xdc,0x24,0x0e,0xe5,0xa9,0xe0,\

 },\

 },\

 {0x00,\

 {\

 0x1b,0x06,0x22,0x30,0x93,0xf3,0xa3,0xb5,0x05,0x00,0x40,0x6e,0x06,0x04,0x07,0x06,0x00,0x21,0x00,0x05,\

 0x02,0xff,0xff,0xff,0xff,0xff,0xff,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x40,0x00,0x2a,0x05,0x01,0x00,0x04,0x04,0x00,0x0a,0x00,0x0b,0x2a,0x27,0x01,0x00,0x80,0x04,\

 0x00,0x0e,0x00,0x00,0x2a,0x29,0x01,0x00,0x80,0x04,0x00,0x10,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x40,0x1c,0x00,0x00,0x2a,0x24,0x01,0x00,0x80,0x04,0x00,0x12,0x00,0x00,0x2a,0x26,0x01,0x00,\

 0x80,0x04,0x00,0x14,0x00,0x00,0x2a,0x50,0x01,0x00,0x80,0x04,\

 },\

 },\

 {0x00,\

 {\

 0x1f,0x06,0x40,0x38,0x00,0x16,0x00,0x00,0x00,0x02,0x02,0x00,0x08,0x04,0x00,0x19,0x00,0x00,0x00,0x03,\

 0x02,0x00,0x02,0x04,0x00,0x1b,0x00,0x1c,0x00,0x04,0x02,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x13,0x06,0x40,0x54,0x0a,0x04,0x00,0x1e,0x00,0x1f,0x00,0x05,0x02,0x00,0x80,0x04,0x00,0x21,0x00,0x00,\

 },\

 },\

 {0x00,\

 {\

 0x13,0x06,0x50,0x00,0x9e,0xca,0xdc,0x24,0x0e,0xe5,0xa9,0xe0,0x93,0xf3,0xa3,0xb5,0x00,0x00,0x40,0x6e,\

 },\

 },\

 {0x00,\

 {\

 0x06,0x06,0xf0,0x00,0x02,0xb2,0xd1,\

 },\

 },\

}

#endif

20. sleepmodeHEADER.h

#ifndef SLEEPMODEHEADER\_H

#define SLEEPMODEHEADER\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#define wdt\_on WDTCONbits.ON

#define wdt\_clear WDTCONbits.WDTCLR

#include <xc.h>

void sleep(void);

void sleep(void)

{

 wdt\_on = 1; // watchdog timer on

 asm("wait"); // go to sleep

 wdt\_on = 0; // watchdoc timer off

}

#ifdef \_\_cplusplus

}

#endif

#endif /\* SLEEPMODEHEADER\_H \*/

21. uart\_over\_ble.h

#ifndef UART\_OVER\_BLE\_H\_\_

#define UART\_OVER\_BLE\_H\_\_

/\*\*

 \* @def UART\_OVER\_BLE\_DISCONNECT

 \* @brief

 \* Command to queue a ACI Disconnect to the nRF8001

 \*/

#define UART\_OVER\_BLE\_DISCONNECT (0x01)

/\*\*

 \* @def UART\_OVER\_BLE\_LINK\_TIMING\_REQ

 \* @brief

 \* Command to queue a ACI Change Timing to the nRF8001

 \*/

#define UART\_OVER\_BLE\_LINK\_TIMING\_REQ (0x02)

/\*\*

 \* @def UART\_OVER\_BLE\_TRANSMIT\_STOP

 \* @brief

 \* Command to stop sending UART over BLE packets

 \*/

#define UART\_OVER\_BLE\_TRANSMIT\_STOP (0x03)

/\*\*

 \* @def UART\_OVER\_BLE\_TRANSMIT\_OK

 \* @brief

 \* Command to allow sending UART over BLE packets

 \*/

#define UART\_OVER\_BLE\_TRANSMIT\_OK (0x04)

typedef struct

{

 uint8\_t uart\_rts\_local; /\* State of the local UART RTS \*/

 uint8\_t uart\_rts\_remote; /\* State of the remote UART RTS \*/

} uart\_over\_ble\_t;

/\*\*

 \* @}

 \*/

#endif // UART\_OVER\_BLE\_H\_\_

22. AMRViewController.h

#import <UIKit/UIKit.h>

#import "BEMSimpleLineGraphView.h"

#import "CoreBluetooth/CBCentralManager.h"

#import "CoreBluetooth/CBPeripheral.h"

#import "CoreBluetooth/CBService.h"

#import "CoreBluetooth/CBCharacteristic.h"

#import "CoreBluetooth/CBUUID.h"

#import "UARTPeripheral.h"

#import "FrontPageViewController.h"

@interface AMRViewController : UIViewController <BEMSimpleLineGraphDataSource, BEMSimpleLineGraphDelegate, UITableViewDataSource, UITableViewDelegate>

@property (weak, nonatomic) IBOutlet BEMSimpleLineGraphView \*myGraph;

@property (strong, nonatomic) NSMutableArray \*arrayOfValues;

@property (strong, nonatomic) NSMutableArray \*arrayOfDates;

@property (strong, nonatomic) NSMutableArray \*arrayOfXLabels;

@property (nonatomic, readwrite) int SkinReader;

@property (strong, nonatomic) UIColor \*colorXaxisLabel;

@property (strong, nonatomic) UIColor \*colorBackgroundYaxis;

@property (weak, nonatomic) IBOutlet UILabel \*GraphTitle;

@property (weak, nonatomic) IBOutlet UIButton \*BluetoothConnector;

@property (weak, nonatomic) IBOutlet UIButton \*TotalValeMEDS;

@property (weak, nonatomic) IBOutlet UILabel \*BluetoothStatus;

@property (weak, nonatomic) IBOutlet UIActivityIndicatorView \*ScanningIndicator;

@property (weak, nonatomic) IBOutlet UILabel \*ExposureValue;

@property (weak, nonatomic) IBOutlet UILabel \*TotalValueLabelMEDS;

//-(NSString \*)labelforXvalueAtIndex:(NSInteger)index;

@end

23. AMRViewController.m

#import "AMRViewController.h"

#import "BEMSimpleLineGraphView.h"

#import "constantsForBLE.h"

@interface AMRViewController () <CBCentralManagerDelegate, CBPeripheralDelegate> {

 int previousStepperValue;

 int totalNumber;

}

@property(nonatomic,strong) CBCentralManager \*centralManager;

@property (retain, nonatomic) NSMutableArray \*connectedServices;

@property (retain, nonatomic) NSMutableArray \*testArrayofValues;

@property (nonatomic, strong) NSMutableArray \*arrayofValues;

@property(nonatomic,strong) CBPeripheral \*mainperipheral;

@property UARTPeripheral \*currentPeripheral;

@property (nonatomic) float sumExposure;

@property (nonatomic) float graphPoints;

@property CBService \*uartService;

@property CBCharacteristic \*rxCharacteristic;

@property CBCharacteristic \*txCharacteristic;

@property FrontPageViewController \*frontPageViewController;

@property bool IntentionalDisconnect;

@property (nonatomic) float SunValueTotaled;

@end

@implementation AMRViewController

@synthesize BluetoothConnector;

@synthesize TotalValeMEDS;

@synthesize TotalValueLabelMEDS;

@synthesize myGraph;

@synthesize BluetoothStatus;

@synthesize ScanningIndicator;

@synthesize GraphTitle;

@synthesize ExposureValue;

@synthesize arrayOfValues;

@synthesize uartService = \_uartService;

@synthesize rxCharacteristic = \_rxCharacteristic;

@synthesize txCharacteristic = \_txCharacteristic;

@synthesize SkinReader;

+ (CBUUID \*) uartServiceUUID

{

 return [CBUUID UUIDWithString:@"6e400001-b5a3-f393-e0a9-e50e24dcca9e"];

}

+ (CBUUID \*) txCharacteristicUUID

{

 return [CBUUID UUIDWithString:@"6e400002-b5a3-f393-e0a9-e50e24dcca9e"];

}

+ (CBUUID \*) rxCharacteristicUUID

{

 return [CBUUID UUIDWithString:@"6e400003-b5a3-f393-e0a9-e50e24dcca9e"];

}

+ (CBUUID \*) deviceInformationServiceUUID

{

 return [CBUUID UUIDWithString:@"180A"];

}

+ (CBUUID \*) hardwareRevisionStringUUID

{

 return [CBUUID UUIDWithString:@"2A27"];

}

#pragma mark - View Lifecycle

- (void)viewDidLoad {

 [super viewDidLoad];

 // Do any additional setup after loading the view, typically from a nib.

 NSLog(@"viewWillAppear");

 ScanningIndicator.hidesWhenStopped=YES;

 BluetoothStatus.text=@"";

 BluetoothStatus.font = [UIFont systemFontOfSize:13];

 // self.graphPoints = 12;

 ExposureValue.text=@"";

 ExposureValue.font = [UIFont systemFontOfSize:13];

 self.graphPoints = 0;

 TotalValueLabelMEDS.text=[NSString stringWithFormat:@"%f",self.graphPoints];

 self.arrayOfValues = [NSMutableArray array];

 if(!self.arrayOfValues) self.arrayOfValues = [[NSMutableArray alloc] init];

 self.arrayOfValues=[[NSMutableArray alloc]initWithObjects:@0,@0, nil];

 GraphTitle.text=@"MEDs vs Time";

 [BluetoothConnector addTarget:self

 action:@selector(scanForBluetooth:)

 forControlEvents:UIControlEventTouchUpInside];

 [TotalValeMEDS addTarget:self action:@selector(ResetTotalValue:) forControlEvents:UIControlEventTouchUpInside];

 self.SunValueTotaled = 0;

 self.sumExposure = 0;

 //self.arrayOfValues = [NSMutableArray array];

 self.arrayOfXLabels = [NSMutableArray array];

 self.testArrayofValues = [NSMutableArray array];

 //if(!self.arrayOfValues) self.arrayOfValues = [[NSMutableArray alloc] init];

 //self.arrayOfValues=[[NSMutableArray alloc]initWithObjects:@0,@0, nil];

 //myGraph.dataSource = self;

// self.myGraph.delegate = self;

 // Create a gradient to apply to the bottom portion of the graph

 CGColorSpaceRef colorspace = CGColorSpaceCreateDeviceRGB();

 size\_t num\_locations = 2;

 CGFloat locations[2] = { 0.0, 1.0 };

 CGFloat components[8] = {

 1.0, 1.0, 1.0, 1.0,

 1.0, 1.0, 1.0, 0.0

 };

 self.myGraph.gradientBottom = CGGradientCreateWithColorComponents(colorspace, components, locations, num\_locations);

 self.myGraph.enableTouchReport = YES;

 self.myGraph.enablePopUpReport = YES;

 self.myGraph.enableYAxisLabel = YES;

 self.myGraph.autoScaleYAxis = YES;

 self.myGraph.alwaysDisplayDots = YES;

 self.myGraph.enableReferenceXAxisLines = YES;

 self.myGraph.enableReferenceYAxisLines = YES;

 self.myGraph.enableReferenceAxisFrame = YES;

 self.myGraph.animationGraphStyle = BEMLineAnimationDraw;

 self.colorXaxisLabel = [UIColor blackColor];

 self.colorBackgroundYaxis = [UIColor blackColor];

 [self hydrateDatasets];

 // Dash the y reference lines

 // Show the y axis values with this format string

}

///////////////////////////////////////////////////////

/\*\*\*\*\*\*\*\*\*\*\*\*\*Button Press\*\*\*\*\*\*\*\*\*\*\*\*/////////////////

///////////////////////////////////////////////////////

#pragma mark - Button Begin

- (IBAction)scanForBluetooth:(id)sender{

 NSLog(@"error here?");

 UIButton \*resultebutton= (UIButton\*)sender;

 NSString \*buttontitle=resultebutton.currentTitle;

 if ([buttontitle isEqual:@"Scan For Bluetooth"]){

 // self.arrayOfValues = [@[] mutableCopy];

 NSLog(@"connectionButtonPressed");

 BluetoothStatus.text=@"Scanning...";

 [ScanningIndicator startAnimating];

 [BluetoothConnector setTitle:@"Stop Scanning" forState:UIControlStateNormal];

 self.IntentionalDisconnect = FALSE;

 self.centralManager = [[CBCentralManager alloc]initWithDelegate:self queue:nil];

 }

 if([buttontitle isEqual:@"Stop Scanning"]){

 BluetoothStatus.text=@"";

 [ScanningIndicator stopAnimating];

 [BluetoothConnector setTitle:@"Scan For Bluetooth" forState:UIControlStateNormal];

 self.IntentionalDisconnect = YES;

 [self.centralManager stopScan];

 }

 if([buttontitle isEqualToString:@"Disconnect"]){

 [BluetoothConnector setTitle:@"Scan For Bluetooth" forState:UIControlStateNormal];

 self.IntentionalDisconnect = YES;

 [self.centralManager cancelPeripheralConnection:self.mainperipheral];

 }

}

-(IBAction)ResetTotalValue:(id)sender{

 [[[UIAlertView alloc] initWithTitle:@"Reset Total" message:@"Clicking this button resets your total exposure to 0 and stores the old value online" delegate:self cancelButtonTitle:@"Cancel" otherButtonTitles:@"OK", nil] show];

}

- (void)alertView:(UIAlertView \*)alertView

clickedButtonAtIndex:(NSInteger)buttonIndex{

 if (buttonIndex == [alertView cancelButtonIndex]) {

 NSLog(@"The cancel button was clicked for alertView");

 }

 else{

 [self.arrayOfValues removeAllObjects];

 self.SunValueTotaled = 0;

 self.graphPoints = 0;

 self.arrayOfValues=[[NSMutableArray alloc]initWithObjects:@0,@0, nil];

 [self hydrateDatasets];

 NSLog(@"graph points = %f",self.graphPoints);

 TotalValueLabelMEDS.text=[NSString stringWithFormat:@"%f",self.graphPoints];

 }

}

//////////////////////////////////////////////

/\*\*\*\*\*\*\*\*\*\*\*\*\*View Controller Enders\*\*\*\*\*\*\*/

/////////////////////////////////////////////

#pragma mark - View Controller Ender

- (void) viewDidUnload

{

 [super viewDidUnload];

}

- (BOOL) shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation

{

 return (interfaceOrientation == UIInterfaceOrientationPortrait);

}

- (void)didReceiveMemoryWarning {

 [super didReceiveMemoryWarning];

 // Dispose of any resources that can be recreated.

}

/////////////////////////////////////////////////////////////////////////////

/\*\*\*\*\*\*\*\*\*\*ALL GRAPH STUFF\*\*\*\*\*\*\*\*/

/////////////////////////////////////////////////////////////////////////////

#pragma mark - Data Correspondance

- (void)hydrateDatasets {

 //if(!self.arrayOfValues) self.arrayOfValues = [[NSMutableArray alloc] init];

 if(!self.arrayOfDates) self.arrayOfDates = [[NSMutableArray alloc] init];

 //[self.arrayOfValues removeAllObjects];

 [self.arrayOfDates removeAllObjects];

 NSLog(@"It is rehydrating %@", self.arrayOfValues);

 [self.myGraph reloadGraph];

 //previousStepperValue = self.graphObjectIncrement.value;

 totalNumber = 0;

 NSDate \*baseDate = [NSDate date];

 BOOL showNullValue = true;

 self.testArrayofValues=[[NSMutableArray alloc]initWithObjects:@0,@0,@0,@7,@5,@3, nil];

 for (int i = 0; i < 9; i++) {

 //float randomValue;

 //randomValue = 0.0;

 //[self.testArrayofValues addObject:@(randomValue)]; // Random values for the graph

 if (i <= 3) {

 [self.arrayOfDates addObject:baseDate]; // Dates for the X-Axis of the graph

 [self labelforXvalueAtIndex:0];

 } else if (showNullValue && i == 4) {

 // [self.arrayOfDates addObject:[self dateForGraphAfterDate:self.arrayOfDates[i-1]]]; // Dates for the X-Axis of the graph

 [self.testArrayofValues addObject:@(1)];

 } else {

 //[self.arrayOfDates addObject:[self dateForGraphAfterDate:self.arrayOfDates[i-1]]];

 // Dates for the X-Axis of the graph

 }

 }

}

/\*- (NSDate \*)dateForGraphAfterDate:(NSDate \*)date {

 NSTimeInterval secondsInTwelveHours = 60;

 NSDate \*newDate = [date dateByAddingTimeInterval:secondsInTwelveHours];

 return newDate;

}

- (NSString \*)labelForDateAtIndex:(NSInteger)index {

 NSDate \*date = self.arrayOfDates[index];

 NSDateFormatter \*df = [[NSDateFormatter alloc] init];

 //df.dateFormat = @"MM/dd";

 NSString \*label = [df stringFromDate:date];

 return label;

}\*/

-(NSString \*)labelforXvalueAtIndex:(NSInteger)index {

 NSInteger \*timeCounter = 0;

 if (self.arrayOfValues.count != 0){

 timeCounter = index;

 }

 NSString \*label2 = [NSString stringWithFormat:@"%d minutes",timeCounter];

 return label2;

}

#pragma mark - SimpleLineGraph Data Source

- (NSInteger)numberOfPointsInLineGraph:(BEMSimpleLineGraphView \*)graph {

 if (self.arrayOfValues.count != 0){

 return (int)[self.arrayOfValues count];

 }

 else{

 //NSLog(@"array of values after method %@", self.arrayOfValues);

 return (int)[self.testArrayofValues count];

 }

}

- (CGFloat)lineGraph:(BEMSimpleLineGraphView \*)graph valueForPointAtIndex:(NSInteger)index {

 if(self.arrayOfValues.count != 0){

 return [[self.arrayOfValues objectAtIndex:index] doubleValue];

 }

 else{

 return [[self.testArrayofValues objectAtIndex:index] doubleValue];

 }

}

#pragma mark - SimpleLineGraph Delegate

- (NSInteger)numberOfGapsBetweenLabelsOnLineGraph:(BEMSimpleLineGraphView \*)graph {

 return 1;

}

- (NSString \*)lineGraph:(BEMSimpleLineGraphView \*)graph labelOnXAxisForIndex:(NSInteger)index {

 //NSString \*label = [self labelForDateAtIndex:index];

 NSString \*label2 = [self labelforXvalueAtIndex:index];

 return [label2 stringByReplacingOccurrencesOfString:@" " withString:@"\n"];

}

- (NSInteger)numberOfYAxisLabelsOnLineGraph:(BEMSimpleLineGraphView \*)graph {

 return 3;

}

- (void)lineGraph:(BEMSimpleLineGraphView \*)graph didTouchGraphWithClosestIndex:(NSInteger)index {

}

- (NSString \*)popUpSuffixForlineGraph:(BEMSimpleLineGraphView \*)graph {

 return @"MEDs";

}

- (void)lineGraph:(BEMSimpleLineGraphView \*)graph didReleaseTouchFromGraphWithClosestIndex:(CGFloat)index {

 //BluetoothStatus.text = @"hey dont touch me!";

 [UIView animateWithDuration:0.2 delay:0 options:UIViewAnimationOptionCurveEaseOut animations:^{

 } completion:^(BOOL finished) {

 }];

}

- (void)lineGraphDidFinishLoading:(BEMSimpleLineGraphView \*)graph {

}

/\* - (void)lineGraphDidFinishDrawing:(BEMSimpleLineGraphView \*)graph {

 // Use this method for tasks after the graph has finished drawing

 } \*/

#pragma mark - We run Bluetooth down here

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Bluetooth Controller \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

-(void)centralManagerDidUpdateState:(CBCentralManager \*)central{

 [self.centralManager scanForPeripheralsWithServices:@[UARTPeripheral.uartServiceUUID] options:@{CBCentralManagerScanOptionAllowDuplicatesKey: [NSNumber numberWithBool:NO]}];

}

- (void)centralManager:(CBCentralManager \*)central

 didDiscoverPeripheral:(CBPeripheral \*)peripheral

 advertisementData:(NSDictionary \*)advertisementData

 RSSI:(NSNumber \*)RSSI {

 [ScanningIndicator stopAnimating];

 NSLog(@"Did discover peripheral %@", peripheral.name);

 [self.centralManager stopScan];

 NSLog(@"we got a hot one");

 NSLog(@"Found peripheral w/ UUID: %@ RSSI: %@ \n AdvData: %@",peripheral.UUID, RSSI,advertisementData);

 [self.centralManager stopScan];

 NSLog(@"Scanning stopped");

 self.mainperipheral = peripheral;

 self.mainperipheral.delegate = self;

 NSLog(@"start connecting to device");

 [self.centralManager connectPeripheral:peripheral options:nil];

}

-(void)centralManager:(CBCentralManager \*)central

 didConnectPeripheral:(CBPeripheral \*)peripheral

{

 NSLog(@"Connected!");

 BluetoothStatus.text=@"Connected!";

 [BluetoothConnector setTitle:@"Disconnect" forState:UIControlStateNormal];

 NSLog(@"scan for services here");

 [self.mainperipheral discoverServices:@[UARTPeripheral.uartServiceUUID]];

}

- (void)peripheral:(CBPeripheral \*)peripheral

didDiscoverServices:(NSError \*)error {

 if (error){

 NSLog(@"Error discovering services: %@", error);

 return;

 }

 for (CBService \*s in [peripheral services])

 {

 if ([s.UUID isEqual:self.class.uartServiceUUID])

 {

 NSLog(@"Found correct service");

 self.uartService = s;

 [self.mainperipheral discoverCharacteristics:@[self.class.txCharacteristicUUID, self.class.rxCharacteristicUUID] forService:self.uartService];

 }

 else if ([s.UUID isEqual:self.class.deviceInformationServiceUUID])

 {

 [self.mainperipheral discoverCharacteristics:@[self.class.hardwareRevisionStringUUID] forService:s];

 }

 }

}

-(void)peripheral:(CBPeripheral \*)peripheral

didDiscoverCharacteristicsForService:(CBService \*)service

 error:(NSError \*)error{

 if (error){

 NSLog(@"Error discovering characteristics: %@", error);

 return;

 }

 NSLog(@"Did discover Charectieristics %@", service.characteristics);

 for (CBCharacteristic \*characteristic in [service characteristics]) {

 if ([characteristic.UUID isEqual:self.class.rxCharacteristicUUID]) {

 // If it is, subscribe to it

 NSLog(@"Found RX characteristic");

 self.rxCharacteristic = characteristic;

 [self.mainperipheral setNotifyValue:YES forCharacteristic:self.rxCharacteristic];

 }

 else if ([characteristic.UUID isEqual:self.class.txCharacteristicUUID]){

 NSLog(@"Found TX characteristic");

 self.txCharacteristic = characteristic;

 }

 else if ([characteristic.UUID isEqual:self.class.hardwareRevisionStringUUID]){

 NSLog(@"Found Hardware Revision String characteristic");

 [self.mainperipheral readValueForCharacteristic:characteristic];

 }

 }

}

- (void)peripheral:(CBPeripheral \*)peripheral didUpdateNotificationStateForCharacteristic:(CBCharacteristic \*)characteristic error:(NSError \*)error{

 if (error) {

 NSLog(@"Error changing notification state: %@", error.localizedDescription);

 }

 // Notification has started

 if (characteristic.isNotifying) {

 NSLog(@"Notification began on %@", characteristic);

 }

}

-(void)peripheral:(CBPeripheral \*)peripheral

didUpdateValueForCharacteristic:(CBCharacteristic \*)characteristic

 error:(NSError \*)error {

 // CSV string with ints as values

 if (error) {

 NSLog(@"Error reading characteristics: %@", [error localizedDescription]);

 return;

 }

 NSLog(@"Received data on a characteristic.");

 if (characteristic == self.rxCharacteristic)

 {

 NSString \*string = [NSString stringWithUTF8String:[[characteristic value] bytes]];

 // [self.mainperipheral didReceiveData:string];

 }

 else if ([characteristic.UUID isEqual:self.class.hardwareRevisionStringUUID])

 {

 NSString \*hwRevision = @"";

 const uint8\_t \*bytes = characteristic.value.bytes;

 for (int i = 0; i < characteristic.value.length; i++)

 {

 NSLog(@"%x", bytes[i]);

 hwRevision = [hwRevision stringByAppendingFormat:@"0x%02x, ", bytes[i]];

 }

 // [self.mainperipheral didReadHardwareRevisionString:[hwRevision substringToIndex:hwRevision.length-2]];

 }

 if (characteristic.value != nil) {

 NSData \*rxData = characteristic.value;

 uint8\_t \*bytes = (uint8\_t \*)[rxData bytes];

 NSString \*SunValueString = [NSString stringWithUTF8String:(char \*)bytes];

 //BluetoothStatus.text= @"You have received this much exposure:";

 //ExposureValue.text= SunValueString;

 NSLog(@"Here is value %s", bytes);

 float SunValueFloat = [SunValueString floatValue];

 float SunValueManipulated = SunValueFloat\*.007;

 NSLog(@"Sun Manipulated : %f", SunValueManipulated);

 self.SunValueTotaled = self.SunValueTotaled + SunValueManipulated;

 NSLog(@"sum exposure = %f", self.SunValueTotaled);

 NSNumber \*SunValueNumber = [[NSNumber alloc] initWithFloat:200.0f];

 SunValueNumber = [NSNumber numberWithFloat: self.SunValueTotaled];

 self.graphPoints = self.graphPoints + SunValueManipulated;

 TotalValueLabelMEDS.text=[NSString stringWithFormat:@"Total: %f",self.SunValueTotaled];

/////////////////////////////////////////////////////////////////////////////////////////////////

 //////////////////Skin Recomendations///////////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////

 if (self.SkinReader == 0){

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You never provided a Skin Type so you cannot receive a recomendation. Click the back button to Choose a Skin Type" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 }

 if (self.SkinReader == 1){

 NSLog(@"SunValue Number %@", SunValueNumber);

 if ([SunValueNumber doubleValue] <= 0.8) {

 BluetoothStatus.text = @"You have received level I exposure";

 ExposureValue.text = @"Enjoy your time in the Sun!";

 }

 else if ([SunValueNumber doubleValue] >= 1.8){

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You have received heavy sun exposure that will cause you to burn. Ensure proper skin protection is used or get out of the sun immediately" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 BluetoothStatus.text = @"You have received level III exposure";

 ExposureValue.text = @"Apply Skin Protection or Get out of the Sun";

 }

 else {

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You have received sun exposure that is starting to cause you to burn. Ensure proper skin protection is used" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 BluetoothStatus.text = @"You have received level II exposure";

 ExposureValue.text = @"Apply Skin Protection and Enjoy your time in the Sun";

 }

 }

 if (self.SkinReader == 2){

 NSLog(@"skin reader: %d", self.SkinReader);

 if ([SunValueNumber doubleValue] <= 1) {

 BluetoothStatus.text = @"You have received level I exposure";

 ExposureValue.text = @"Enjoy your time in the Sun!";

 }

 else if ([SunValueNumber doubleValue]>= 2.8){

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You have received heavy sun exposure that will cause you to burn. Ensure proper skin protection is used or get out of the sun immediately" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 BluetoothStatus.text = @"You have received level III exposure";

 ExposureValue.text = @"Apply Skin Protection or Get out of the Sun";

 }

 else {

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You have received sun exposure that is starting to cause you to burn. Ensure proper skin protection is used" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 BluetoothStatus.text = @"You have received level II exposure";

 ExposureValue.text = @"Apply Skin Protection and Enjoy your time in the Sun";

 }

 }

 if (self.SkinReader == 3){

 NSLog(@"skin reader: %d", self.SkinReader);

 if ([SunValueNumber doubleValue]<= 1.6) {

 BluetoothStatus.text = @"You have received level I exposure";

 ExposureValue.text = @"Enjoy your time in the Sun!";

 }

 else if ([SunValueNumber doubleValue]>= 4.6){

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You have received heavy sun exposure that will cause you to burn. Ensure proper skin protection is used or get out of the sun immediately" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 BluetoothStatus.text = @"You have received level III exposure";

 ExposureValue.text = @"Apply Skin Protection or Get out of the Sun";

 }

 else {

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You have received sun exposure that is starting to cause you to burn. Ensure proper skin protection is used" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 BluetoothStatus.text = @"You have received level II exposure";

 ExposureValue.text = @"Apply Skin Protection and Enjoy your time in the Sun";

 }

 }

 if (self.SkinReader == 4){

 NSLog(@"skin reader: %d", self.SkinReader);

 if ([SunValueNumber doubleValue]<= 2.1) {

 BluetoothStatus.text = @"You have received level I exposure";

 ExposureValue.text = @"Enjoy your time in the Sun!";

 }

 else if ([SunValueNumber doubleValue]>= 6){

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You have received heavy sun exposure that will cause you to burn. Ensure proper skin protection is used or get out of the sun immediately" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 BluetoothStatus.text = @"You have received level III exposure";

 ExposureValue.text = @"Apply Skin Protection or Get out of the Sun";

 }

 else {

 [[[UIAlertView alloc] initWithTitle:@"Warning" message:@"You have received sun exposure that is starting to cause you to burn. Ensure proper skin protection is used" delegate:nil cancelButtonTitle:@"I understand" otherButtonTitles:nil] show];

 BluetoothStatus.text = @"You have received level II exposure";

 ExposureValue.text = @"Apply Skin Protection and Enjoy your time in the Sun";

 }

 }

/////////////////////////////////////////////////////////////////////////////////////////

 /////////ADD Value to Graph/////////////////////

////////////////////////////////////////////////////////////////////////////////////////

 //[self.arrayOfValues addObject:@(SunValueInt)];

 [self.arrayOfValues addObject:SunValueNumber];

 [self hydrateDatasets];

 [self.centralManager cancelPeripheralConnection:self.mainperipheral];

 }

}

- (void)centralManager:(CBCentralManager \*)central

didDisconnectPeripheral:(CBPeripheral \*)peripheral

 error:(NSError \*)error{

 if (self.IntentionalDisconnect == NO) {

 NSLog(@"Connecting Again");

 self.centralManager = [[CBCentralManager alloc]initWithDelegate:self queue:nil];

 }

 else{

 NSLog(@"Officially disconnected");

 ExposureValue.text=@"";

 BluetoothStatus.text=@"";

 [BluetoothConnector setTitle:@"Scan For Bluetooth" forState:UIControlStateNormal];

 }

}

 /\* NSLog(@"Here I am");

 NSArray \* dataArray = [[[NSString alloc] initWithData:characteristic.value

 encoding:NSUTF8StringEncoding] componentsSeparatedByString:@", "];

 NSMutableArray \* numberArray = [@[] mutableCopy];

 for (NSString \*numberString in dataArray) {

 [numberArray addObject:@([numberString intValue])];

 }

 NSLog(@"This is the Number Array %@", numberArray);

 // do stuff with numberArray

 self.sumExposure = 0;

 int i = self.sumExposure;

 self.sumExposure += 7;//persistently saves and adds 7 to value

}\*/

/\*

 \* Getters and Setters

 \*/

-(float)graphPoints //Getter

{

 return [[[NSUserDefaults standardUserDefaults] objectForKey:@"GRAPH POINTS"] intValue];

}

-(void)setGraphPoints:(float)graphPoints //SETTER

 {

 [[NSUserDefaults standardUserDefaults] setObject:@(graphPoints)

 forKey:@"GRAPH POINTS"];

 }

-(float)sumExposure //GETTER

{

 return [[[NSUserDefaults standardUserDefaults] objectForKey:@"SUM EXPOSURE"] intValue];

}

-(void)setSumExposure:(float)sumExposure //SETTER

{

 [[NSUserDefaults standardUserDefaults] setObject:@(sumExposure)

 forKey:@"SUM EXPOSURE"];

}

//to access sumExposure use self.sum....

@end