802.15.4 MAC API Programming Example Guide

1. Introduction

The 802.15.4/ZigBee™ Development Board can be used for demo purposes or as a platform for firmware development. This document describes the MAC Blinky code example using the 802.15.4 MAC layer Application Programming Interface (API). The MAC Blinky code example demonstrates association, data transmission, and disassociation. This code example may be used as a starting point to develop a 802.15.4 MAC application.

The 802.15.4/ZigBee Development Board included in the 802.15.4 Development kit comes with firmware installed for a 802.15.4 demo. To use the 802.15.4/ZigBee Development Board for code development, you will have to erase the existing firmware. If desired the demo code can later be restored by downloading the appropriate hex file.

The 802.15.4 Development kit includes two 802.15.4/ZigBee development boards. These two boards can be used with the MAC Blinky software to demonstrate a point-to-point network. Additional 802.15.4/ZigBee development boards may be purchased and used to create a point to multi-point or star network

Refer to “AN222: 2.4 GHz 802.15.4/ZigBee Development Board Hardware User's Guide” for schematics, PCB layout, and configuration instructions. Refer to “AN268: 802.15.4 MAC Application Programming Interface Layer Guide” for a complete description of the 802.15.4 API commands and usage. The IEEE 802.15.4-2003 specification can be obtained from http://standards.ieee.org. For a limited time, the standard is available free of charge from the Get IEEE 802 ® program at http://standards.ieee.org/getieee802.

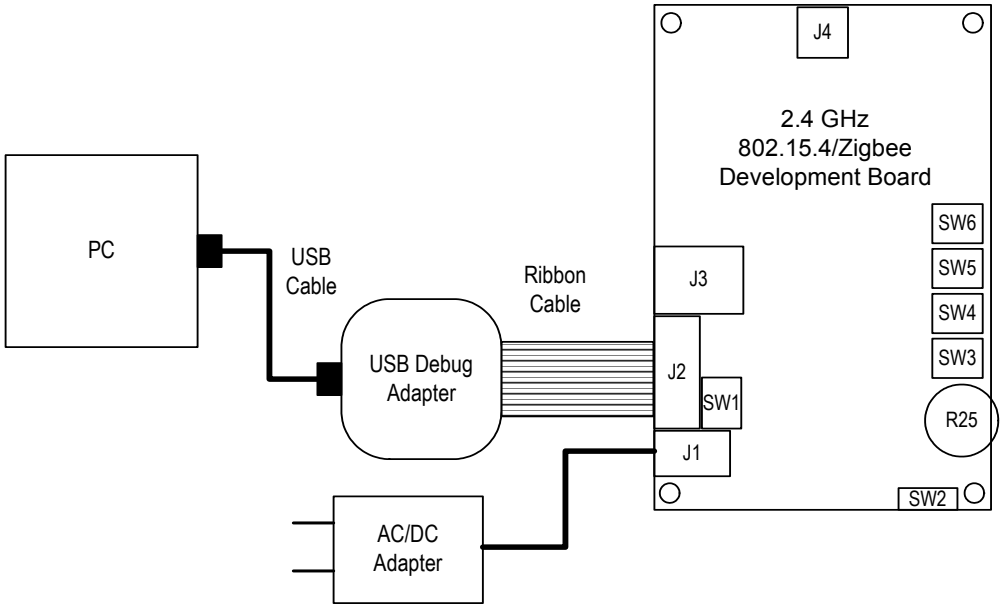

Figure 1. Programming Setup

2. Connecting to the IDE

The 802.15.4/ZigBee development board can be used with the Silicon Laboratories Integrated Development Environment (IDE) and the Universal Serial Bus (USB) Debug Adapter as shown in Figure 1.

1. Connect the USB Debug adapter to the PC.
2. Connect the ten pin ribbon cable to the 802.15.4/ZigBee development board (J2).
3. Connect the 9 V universal ac/dc adapter to the 2.1 mm power supply jack (J1) on the 802.15.4/Zigbee Development Board.

3. Downloading the Firmware

1. Launch the Silicon Laboratories IDE.
2. From the Options menu select the connection options sub-menu.
3. Choose the USB Debug adapter if you are using the USB Debug Adapter or RS232 Serial Adapter.
4. Select the JTAG radio button for the Debug interface. Click OK to close the dialog box.
5. From the Debug menu, select Connect or click on the connect icon in the toolbar.

The status bar on the bottom of the IDE window should display the connection status “Target:C8051F121”. This indicates that the debug adapter has successfully connected to the C8051F121 and is ready to download code.

The Intel hex file for the demo firmware is located in the following directory:

```
\SiLabs\MCU\Examples\802_15_4\MAC_Blinky\output\
```

To download the firmware select “download object file...” from the debug menu. Do **not** Erase the entire code space. This will erase the 64-bit Extended Unique Identifier stored in Flash memory.

Click on “Browse…” In the browse dialog box, find the “Files to List” pulldown menu and select “Intel - Hex” Then browse to the location of the MAC_Blinky.hex file. The demo file will be downloaded into Flash.

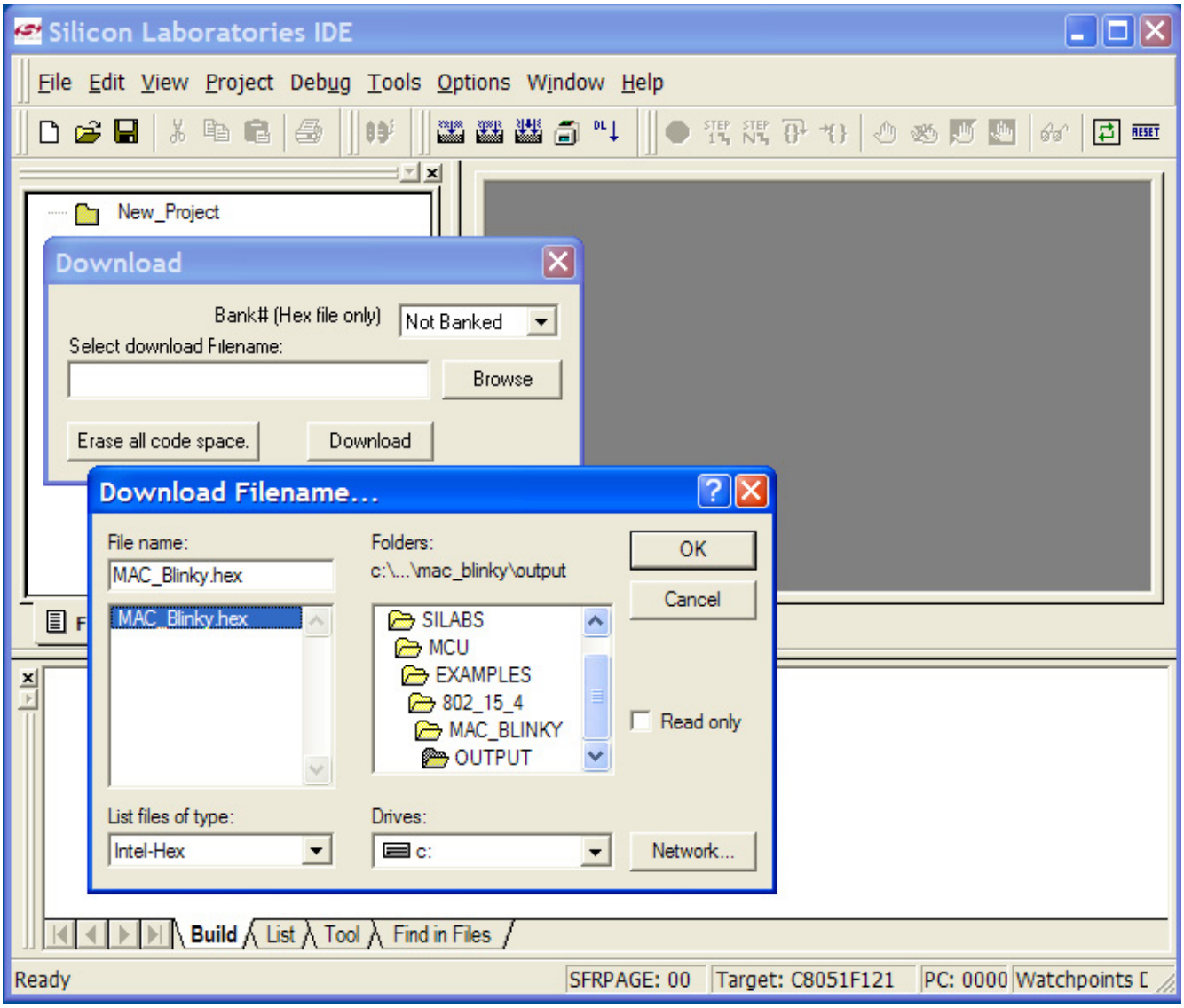

Figure 2. Downloading Firmware

Select “Go” from the debug menu or click on the green Go button in the toolbar to run the program. Three of the colored LEDs on the board should blink. This indicates the code has been downloaded and is functional. Click on the red stop button to halt execution.

Select “Disconnect” from the Debug menu. Now, unplug the power connection and 10-pin header. Repeat this procedure for each board you wish to reprogram.

If you inadvertently erase the EUI you can modify the EUI directly from the IDE. Connect to the 802.15.4/ZigBee development board. Select “Code Memory” from the View→Debug Windows menu. Type “1FBF6” in the code address box. Click in the code memory window just left of the first byte of address FBF6. Enter the EUI from the sticker on the back of the board in little endian format. The least significant byte of the EUI should be in location 0x1FBF6 and the most significant byte should be in location 0x1FBFF.

4. MAC Blinky Demo Functional Description

The MAC Blink demo demonstrates a basic 802.15.4 network. The 802.15.4 Development kit comes with two 802.15.4/ZigBee development boards. These two boards can be used with the MAC Blinky software to demonstrate a point-to-point network. Additional 802.15.4/ZigBee development boards may be purchased individually and used to create a point to multi-point or star network.

Select one of the 802.15.4/ZigBee development boards to be the 802.15.4 PAN Coordinator. Push button SW6 to configure the board as a PAN Coordinator and start the network. The green LED on the Coordinator will turn on.

The PAN Coordinator is responsible for initiating the entire personal area network (PAN). The PAN Coordinator selects the channel and the 16-bit PAN ID that will be used for the entire PAN. In this demonstration the PAN ID and frequency are defined as macro constants. A Coordinator is any Device that permits other Devices to associate. Once a Device associates, it can then send and receive data to the Coordinator.

The second board is designated as an 802.15.4 End Device. An End Device is any 802.15.4 Device that is not a Coordinator. End Devices can associate with a Coordinator, but do not permit other Devices to associate.

Pushing button SW5 on the End Device board will initiate the process for joining a network. First the Device performs an active scan. The active scan will scan all channels to locate potential Coordinators. If the Device locates a suitable Coordinator, it will attempt to associate. If the association is successful, the yellow LED on the Coordinator will blink once and the yellow LED on the Device will turn on.

Pushing button SW4 on the End Device will send data to the Coordinator. The red LED will blink once on both the End Device and the Coordinator.

Pushing button SW3 on the End Device will cause the Device to disassociate from the network. The amber LED on the Coordinator will blink once when a Device disassociates. The yellow LED on the End Device will turn off when it disassociates.

	Coordinator Functions	End Device Functions	MAC Primitives
SW6	Start as PAN Coordinator		MLME_Reset.request MLME_Set.request MLME_Start.request
SW5	N/A	Associate with Coordinator	MLME_Scan.request MLME_Associate Request
SW4	N/A	Send Data to Coordinator	MCPS_Data.request
SW3	N/A,	Disassociate	MLME_Disassociate.request

Table 1. Push Button Functions

5. Rebuilding the Project

The source code for the MAC Blinky demo is included with the development kit. The project folder includes source files for the essential main and MAC_Blinky modules, a library file which contains the 802.15.4 MAC code, and all of the header files required to rebuild the project. The project also contains a Silicon Labs IDE workspace file that can be used to edit and build the file.

The library file is a special library to facilitate building the project with the demo tools. The project can be rebuilt using the included demonstration tools with a 4 kB linker limit. The library itself does not count against the 4 kB linker limit. Thus, it is possible to rebuild the project using the demo tools. The demo tools are for evaluation purposes only. The user is limited to non-commercial use only as specified in the license agreement. The user will have to purchase a full set of development tools for commercial development.

5.1. Rebuilding the MAC Blinky Project

1. Connect the 802.15.4/ZigBee Development Board as shown in Figure 1.
2. Connect the USB Debug adapter or the Serial Adapter or to the PC.
3. Connect the ten pin ribbon cable to the 802.15.4/ZigBee development board.
4. Connect the 9 V AC/DC Adapter to the 2.1 mm power supply jack on the 802.15.4 Development Board.
5. Launch the Silicon Laboratories IDE.
6. From the “Project” menu select “Open Project...”.
7. Browse to the MAC_Blinky folder:
   SiLabs\MCU\Examples\802_15_4\MAC_Blinky
8. Open the workspace file, “MAC_Blinky.wsp”.
9. From the File View tab, double-click on the source file. *main.c*.
   This will open the application source window in the editor pane. Expand this window to fill the middle pane.
10. From the Debug menu, select “Rebuild Project”.
    The IDE will compile and link all files in the project. The IDE should display the linker status as shown in Figure 3.

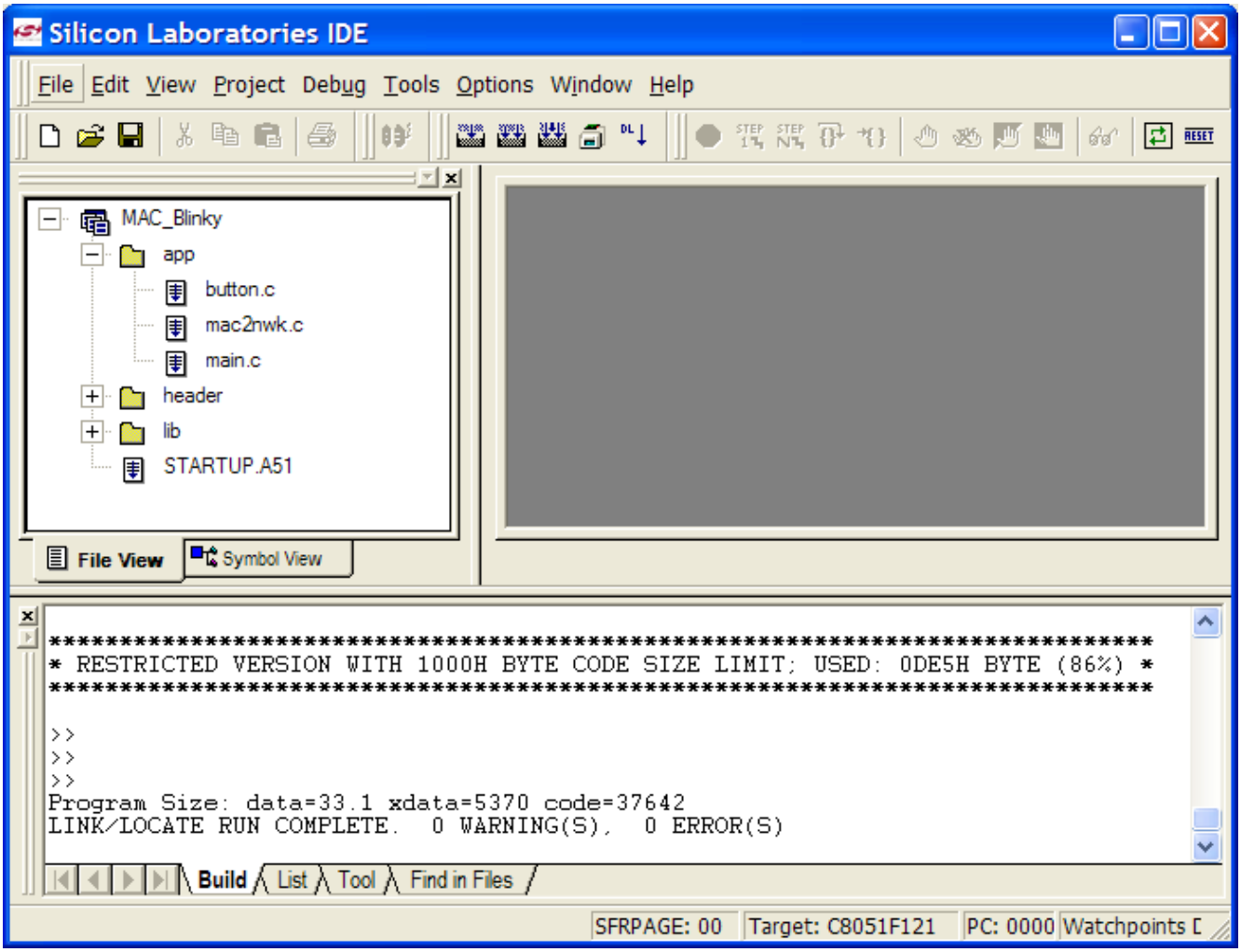

Figure 3. Build Complete, No Errors

5.2. Debug Demonstration

1. Select “Connect” from the “Debug” menu. The status bar on the bottom of the IDE window should display the connection status "Target:C8051F121".
2. Then select “Download” also from the debug menu.
   The IDE will download the code to the target board.
3. From the File View tab, double-click on the source file. *main.c*.
   This will open the *main.c* source window in the editor pane.
4. Scroll down in the *main.c* window to locate the *main()* function. Locate the first line of code in the main function. Right click on this line and chose Insert-Breakpoint from the pop-up menu.
5. Click on the green Go button or select Go from the debug window. Code execution will stop at the breakpoint as shown in Figure 4

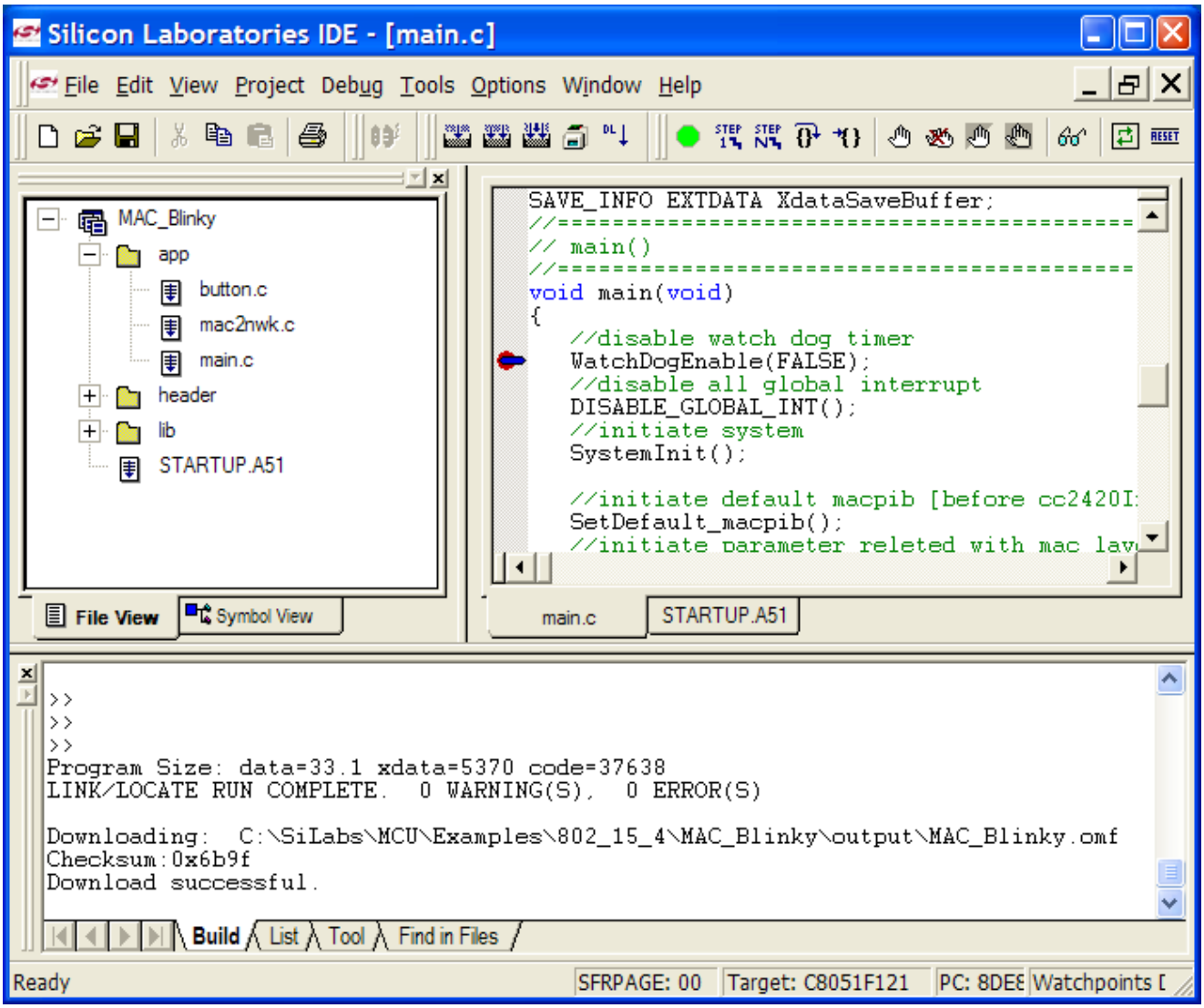

Figure 4. Run to Main

6. MAC Blinky 802.15.4 Primitive Usage

This section describes the MAC Blinky demo code operation in detail using the 802.15.4 primitives. This requires a working knowledge of the 802.15.4 specification. In particular, you should familiarize yourself with the terms and definitions defined in 802.15.4.

The implementation is explained using code excerpts. These code excerpts were taken from the source code and simplified in some cases for illustration purposes. The MAC interface is explained in detail in AN268.

Pushing button SW6 will configure the board as a PAN Coordinator and start the network. This process requires four primitives. First the MAC layer is reset using the MLME-Reset.request primitive. The SetDefaultPIB parameter is set to TRUE. This ensures that the MAC is reset and ready to start as a PAN Coordinator from any state.

```
n_m_msg_t.msg.MLME_RESET_request.setDefaultPib = TRUE;
n_m_msg_t.primtype=gMlmeResetReq_c|NWK_Buffer_Valid;
```

Next the macShortAddress is set to 0x0000 using the MLME-Set.request primitive. This step is required before

calling the MLME_Start.request primitive. The MLME-Start.request primitive will return an error if the short address has not been set first. The 802.15.4 specification does not specify a preferred short address for the PAN Coordinator. However, the ZigBee NWK specification does specify 0x0000 for the PAN Coordinator and this convention has been adopted for the demo.

```
n_m_msg_t.msg.MLME_SET_request.pibAttribute = macShortAddressID;
n_m_msg_t.msg.MLME_SET_request.pibAttributeValue = tmpShortAddress;
n_m_msg_t.primtype = gMlmeSetReq_c|NWK_Buffer_Valid;
```

The network is then started using the MLME-Start.request primitive. The macPANId parameter is set to the DEMO_PAN_ID value. The LogicalChannel is set to the DEMO_CHANNEL value. These values are defined using macros for the demo. The beaconOrder and superframeOrder are set to 15 for a non-beacon PAN. The panCoordinator parameter is set to TRUE. The Coordinator alignment parameter is set to TRUE, configuring the PAN Coordinator to send a Coordinator realignment frame to announce its presence on the channel. The Coordinator alignment is not essential, but does aid in network debugging when using a sniffer

```
n_m_msg_t.msg.MLME_START_request.panId = DEMO_PAN_ID;
n_m_msg_t.msg.MLME_START_request.logicalChannel = DEMO_CHANNEL;
n_m_msg_t.msg.MLME_START_request.beaconOrder = 0x0F;
n_m_msg_t.msg.MLME_START_request.superFrameOrder = 0x0F;
n_m_msg_t.msg.MLME_START_request.panCoordinator = TRUE;
n_m_msg_t.msg.MLME_START_request.batteryLifeExt = FALSE;
n_m_msg_t.msg.MLME_START_request.coordRealignment = TRUE;
n_m_msg_t.msg.MLME_START_request.securityEnable = FALSE;
n_m_msg_t.primtype = gMlmeStartReq_c|NWK_Buffer_Valid;
```

If the MLME-Start.confirm primitive returns a status of SUCCESS, the green LED D6 will be turned on.

Lastly the MLME-Set.request primitive is used to set macAssociatePermit to TRUE. This last step is essential as the default value is FALSE. This should be done after the MLME-Start primitive. Setting macAssociatePermit to TRUE will set the permit association bit in the beacon frame and allow Devices to associate with the Coordinator.

```
n_m_msg_t.msg.MLME_SET_request.pibAttribute = macAssociationPermitID;
n_m_msg_t.msg.MLME_SET_request.pibAttributeValue = TRUE;
n_m_msg_t.primtype = gMlmeSetReq_c|NWK_Buffer_Valid;
```

Pushing button SW5 on and End Device will initiate the process of joining a network.The End Device will first reset the MAC using the MLME-Reset.request primitive. This step ensures that the MAC is in a known state before association.

```
n_m_msg_t.msg.MLME_RESET_request.setDefaultPib=TRUE;
n_m_msg_t.primtype=gMlmeResetReq_c|NWK_Buffer_Valid;
```

Next the End Device will call the MLME-Scan.request primitive to initiate an active scan. The scanChannels parameter is set to scan all of the 2.4 GHz channels.

```
n_m_msg_t.msg.MLME_SCAN_request.scanChannels = 0x07fff800L;
n_m_msg_t.msg.MLME_SCAN_request.scanType = ACTIVE_SCAN;
```

```
n_m_msg_t.msg.MLME_SCAN_request.scanDuration = 0x03;
n_m_msg_t.primtype = gMlmeScanReq_c|NWK_Buffer_Valid;
```

Upon completion of the scan, the MAC will call the MLME-scan.confirm primitive. The demo application will check the results of the scan parameters. First the application checks to make sure the resultListSize are not zero. If the resultListSize is not zero, the application will check the PanDescriptorList of each Coordinator found during the active scan. This demo application will use the first Coordinator it finds that has a CoordinatorPANId matching the DEMO_PAN_ID. A more complex example might also consider the Link Quality and other criteria from the PanDescriptorList.

If the application finds a suitable Coordinator, it will call the MLME-Associate.request primitive. Many of the parameters are set according to the PAN descriptor for the selected Coordinator. The PANId parameter will be set to the CoordinatorPANId. The logicalChannel will be set to the channel of the Coordinator. The coordAddrMode and coordAddress are set according to the address of the Coordinator. The capabilityInfo parameter is set to 0x80, the msb indicating that the Allocate Address bit is set to 1.

```
n_m_msg_t.msg.MLME_ASSOCIATE_request.logicalChannel = LogicalChannel;
n_m_msg_t.msg.MLME_ASSOCIATE_request.coordAddrMode = CoordAddrMode;
n_m_msg_t.msg.MLME_ASSOCIATE_request.coordPanId = PANId;
for(i = 0;i<8;i++)
{
   n_m_msg_t.msg.MLME_ASSOCIATE_request.coordAddress[i] = CoordAddress[i]
}
n_m_msg_t.msg.MLME_ASSOCIATE_request.capabilityInfo = 0x80;
n_m_msg_t.msg.MLME_ASSOCIATE_request.SecurityLevel = 0x00;
n_m_msg_t.primtype = gMlmeAssociateReq_c|NWK_Buffer_Valid;
```

When the PAN Coordinator receives an association request, the Coordinator MAC will call the MLME-Associate.indication primitive. The application will then allocate a sequential number for the assocShortAddress and call the MLME-Associate.response primitive. The application will also blink the yellow LED D7. After allocating eight addresses, the application will refuse further associations and respond with the status PAN at Capacity.

```
n_m_msg_t.msg.MLME_ASSOCIATE_response.assocShortAddress[0] = AssociatedAddress[0];
n_m_msg_t.msg.MLME_ASSOCIATE_response.assocShortAddress[1] = AssociatedAddress[1];
n_m_msg_t.msg.MLME_ASSOCIATE_response.status = 0x00;
n_m_msg_t.primtype = gMlmeAssociateRes_c|NWK_Buffer_Valid;
```

Upon completion of the association procedure, the MAC of the End Device will call the MLME-Associate.confirm primitive. If the confirm status equals SUCCESS, the yellow LED D7 on the End Device will be illuminated continuously.

Pushing SW4 on the End Device will send data to the Coordinator. The data is sent to the Coordinator using the MCPS-Data.request primitive. The destination address parameter dstAddr is set to the address of the Coordinator. The source address srcAddr parameter is set to macShortAddress. The source and destination PANId parameter are set to macPANId. The value of txOptions parameter is set to 0x01, the lsb is set to 1 for acknowledged transmission.

```
n_m_msg_t.msg.MCPS_DATA_request.dstAddr[0] = macCoordShortAddress[0];
n_m_msg_t.msg.MCPS_DATA_request.dstAddr[1] = macCoordShortAddress[1];
```

```
n_m_msg_t.msg.MCPS_DATA_request.srcAddrMode = srcSHORT_ADDRESS_MODE;
n_m_msg_t.msg.MCPS_DATA_request.dstAddrMode = dstSHORT_ADDRESS_MODE;
n_m_msg_t.msg.MCPS_DATA_request.srcPanId    = macPANId;
n_m_msg_t.msg.MCPS_DATA_request.srcAddr[0]  = macShortAddress[0];
n_m_msg_t.msg.MCPS_DATA_request.srcAddr[0]  = macShortAddress[1];
n_m_msg_t.msg.MCPS_DATA_request.dstPanId    = macPANId;
n_m_msg_t.msg.MCPS_DATA_request.msduHandle  = 0x01;
n_m_msg_t.msg.MCPS_DATA_request.txOptions   = 0x01;
n_m_msg_t.msg.MCPS_DATA_request.msduLength  = 16;
for(i=0;i<16;i++)
{
   n_m_msg_t.msg.MCPS_DATA_request.msdu[i]=i;
}
n_m_msg_t.primtype = gMcpsDataCnf_c|NWK_Buffer_Valid;
```

Upon the successful reception of data the Coordinator MAC will call the MCPS-Data.indication primitive. This will blink the red LED D9 on the Coordinator board.

Upon the acknowledged transmission, the MAC of the Device will call the MCPS-Data.confirm primitive. This will blink the red LED D9 on the End Device board.

Pressing push-button SW4 on the End Device board will initiate disassociation. The application will call the MLME-Disassociate.request primitive. The destination DeviceAddress is set to the macCoordExtendedAddress. The value of disassociateReason parameter is set to 0x02, Device wishes to Disassociate.

```
n_m_msg_t.msg.MLME_DISASSOCIATE_request.DisassShortAddress=0xFFFE;
for(i=0;i<8;i++)
{
   n_m_msg_t.msg.MLME_DISASSOCIATE_request.deviceAddress[i]= macCoordAddress[i];
}
n_m_msg_t.msg.MLME_DISASSOCIATE_request.disassociateReason= 0x02;
n_m_msg_t.primtype = gMlmeDisassociateReq_c|NWK_Buffer_Valid;
```

This implementation includes an additional parameter DisassShortAddress, to permit the use of a short address for the MLME-Disassociate.request. The 802.15.4 standard specifies that a disassociate request should always use extended addressing for both source and destination. When the Coordinator wishes the Device to associate, it must use the indirect method of data transmission. However, the associated Device might only request data using the short address. In this case you must use the short address of the associated Device. In all other cases, set DisassShortAddress to 0xFFFE indicating the MAC should use the extended address for the disassociate request.

When the Coordinator receives a disassociate notification frame it will call the MLME-Disassociate.indication primitive. This will blink the amber LED D8 on the Coordinator.

Upon acknowledge transmission of the disassociate notify frame, the MAC of the End Device will call the MLME-Disassociate.confirm primitive. If the confirm status equals SUCCESS, the end Device will extinguish yellow LED D7.

7. Using the MAC Blinky Demo with a Packet Sniffer

Using the MAC Blinky demo with an 802.15.4 packet analyzer or sniffer is highly recommended. The ability to observe 802.15.4 activity over the air and decode the 802.15.4 packets is essential to develop 802.15.4 applications.

A low cost packet sniffer is available from Chipcon with the CC2420EB evaluation board. This sniffer software works well with the CC2420 chip and provides decoding of the 802.15.4 MAC layer. The Chipcon sniffer software was used for the packet captures in this application note.

The MLME-Start primitive optionally sends out a Coordinator realignment frame. The realignment frame includes information about the current channel and PAN ID. The Coordinator realignment frame tells us the Coordinator is up and running.

The active scan will send a beacon request on all 16 channels and waits for a response. The PAN Coordinator will be listening on one of these channels and will send a beacon in response to the request.

The association transaction consists of an association request and data request from the Device wishing to associate followed by an association response from the Coordinator. Each of these three frames are sent with the acknowledge requested bit set and should be followed with a corresponding ACK frame. A packet sniffer capture from a successful start, scan, and association is shown in Figure 5.

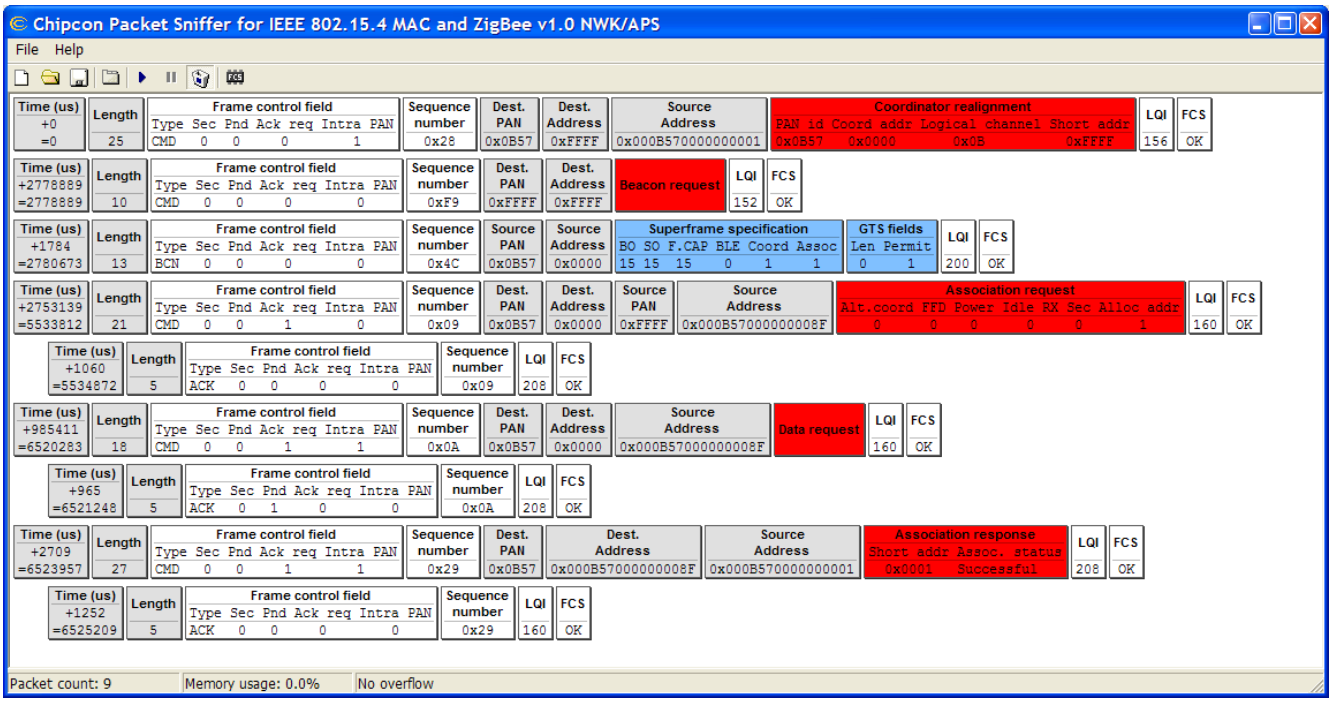

Figure 5. Association Transaction

Direct data is transmitted from the End Device to the PAN Coordinator. The End Devices in a non-beacon network might send data at any time.The PAN Coordinator will respond with an ACK frame. A packet sniffer capture for direct data transmission is shown in Figure 6

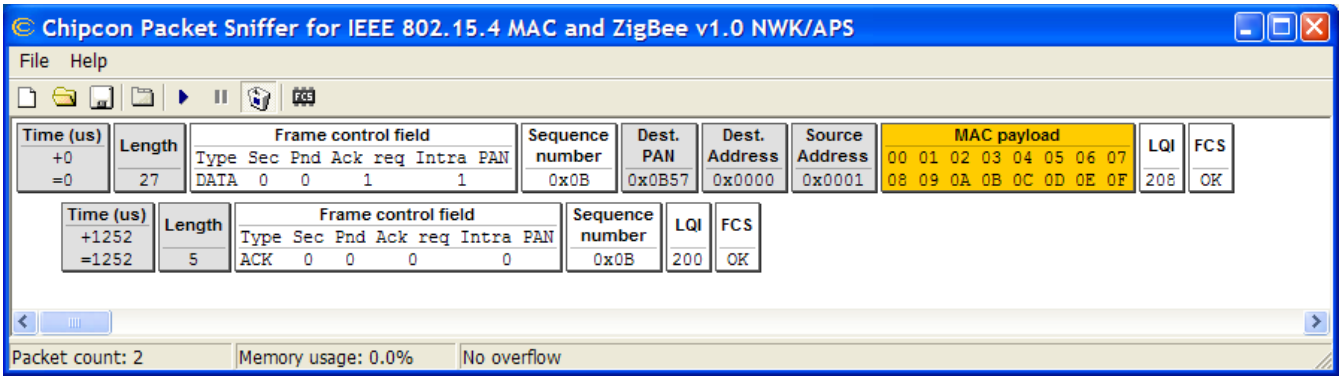

Figure 6. Data Transmission

Disassociation is accomplished when the End Device sends a disassociate notify frame to the PAN Coordinator. This frame is acknowledged, but there is no response. A packet sniffer capture for disassociation is shown in Figure 7

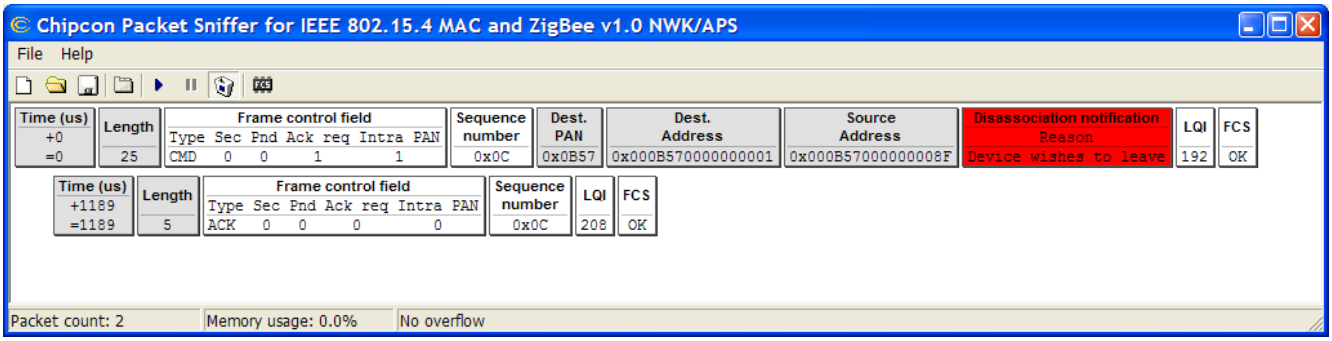

Figure 7. Disassociation

If the PAN Coordinator already has eight associated Devices, it will deny association and respond with status of PAN at capacity. A packet sniffer capture an unsuccessful association transaction is shown in Figure 8. In this case the Associate Short Address field will be set to 0xFFFF.

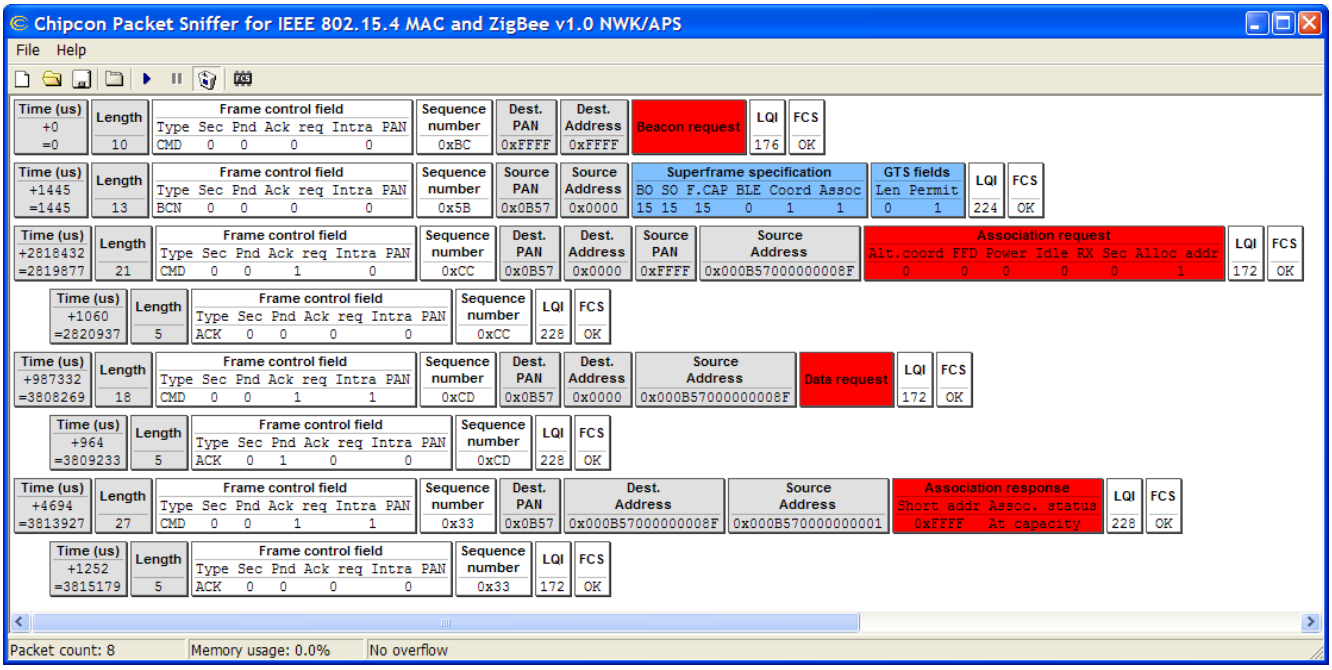

Figure 8. PAN at Capacity

8. MAC Blinky Code Details

8.1. Main

The main module *main.c* includes the *main()* function called upon startup. The main function first calls several initialization routines and then the main loop. Note that the initialization and many of the internal functions are specific to the C8051F121 hardware.

The main loop for the code uses round-robin scheduling. Any task in the main loop should be written using non-blocking code. Each process will make a best-effort to perform the required task. If resources are not ready to perform the task, the task should be deferred until the next iteration of the main loop. Some tasks may require that the mac is ready to receive a new MAC primitive.

If you add anything to the main loop it should also be implemented as non-blocking code.

8.2. Button

The *button.c* module includes code for checking the buttons and performing specific tasks to service the buttons.

The *CheckButton()* function polls and debounces the four buttons. Each button has an associate count variable in the *Button[]* array. The upper two bits are used as state variables for the state of the corresponding button. This function is called from main and is non-blocking code. The debounce function requires that the button be held down for greater than *MAXCOUNT* iterations and also released for the same period.

Once a button press has been detected, the corresponding *Button[]* value will be set to *BUTTON_PRESS*. The corresponding *Button[]* value is reset to *BUTTON_UP* status by the associate button function once completely serviced.

The *ProcessButton()* function processes the functions associated with each button. Since this function is called from *main()* it is a non- blocking function. The buttons will only be processed if the MAC is ready for a new command. If the Network to MAC buffer has a MAC task pending, the *ProcessButton()* function will not process the button commands in this iteration of the main loop. Some of the button functions have dependencies or pre-existing conditions. Upon reset the Device may be configured as a Coordinator by pressing *Button[0]* or can associate with a Coordinator by pressing *Button[1]*. After this, *Button[0]* and *Button[1]* should be ignored. The *macShortAddress* is set to *0XFFFF* upon reset. The short address is set to *0x0000* for the Coordinator or is set to some number less than *0xFFFE* upon association. Thus, the short address may be used to test for Coordinator or association.

The *buttonStartRequest()* function is called from *ProcButton()* when the start button has been pressed. The *ProcButton()* function checks for an empty Network to MAC buffer and this function assumes the buffer is empty. It takes four MAC primitives to reset and configure the Coordinator. So a state machine is used in this function. The function will continue to be called until the button status is cleared by the final state.

The *ButtonAssociateRequest()* function is called from the *ProcButton()* function when the associate button has been pressed. The *ProcButton()* function checks for an empty network to mac buffer and this function assumes the buffer is empty. This function contains a state machine with two states. The first state resets the MAC. The second state initiates an active scan. The rest of the association transaction is handled by the mac2nwk callback functions. Upon conclusion of the active scan, a scan confirm will be sent to the network layer. The scan confirm will initiate an association request. If the association confirm is successful, an LED will be turned on.

The *buttonDataRequest()* function is called from *ProcButton()* when the data request button has been pressed. The *ProcButton()* checks for an empty Network to MAC buffer and this function assumes the buffer is empty. This function will send data to the Coordinator using the short address of the Coordinator as the destination and the associated short address as the source. Dummy data is used just to demonstrate data transmission. In a typical application this data might be an ADC measurement.

The *buttonDisassociateRequest()* function is called from *ProcButton()* when the disassociate button has been pressed. The *ProcButton()* checks for an empty network to mac buffer and this function assumes the buffer is empty. Once the disassociation notify frame has been acknowledged, a disassociate confirm will be sent to the network layer. If the disassociate confirm is successful, an LED will be turned off.

8.3. MAC2NWK

The *MAC2NWK.c* module contains functions related to MAC primitives that are initiated from the mac and sent to the network NWK or next higher level. This includes indication and confirm primitive types.

The *ProcessMAC2NWK()* first checks to see if there is anything in the MAC to network message buffer, m2n_msg. Because the MAC to network buffer has multiple entries, a for loop is used to check the *NWK_BUFFER_VALID* bit of each entry. The *ProcessMAC2NWK()* function will only process the first primitive it finds. If there are multiple entries, only one will be processed.

Normally the MAC to NWK message is processed and the *NWK_BUFFER_VALID* is cleared. However, in a few cases the message processing must be deferred until the next iteration. Some MAC to NWK messages will initiate additional network to MAC messages. If the network to mac buffer is full, the message cannot be processed and is deferred until the next iteration.

A large case statement is used to process the MAC to network message. Most cases have simple actions such as blinking or changing the state of an LED. The cases which require complex actions are documented below.

Mac to network case *gNwkScanCnf_c* corresponds to the MLME-scan.confirm primitive. First the result list size is checked. If the results list is non-zero, then the PAN descriptor is checked for each Coordinator found. The first Coordinator matching the DEMO_PAN_ID is used for association. If the network to mac buffer is empty, an MLME-Association.request will be issued. If the buffer is not ready to receive a new message, processing will be deferred.

Case *gNwkAssociateInd_c* corresponds to the MLME.Associate.indication primitive. The network or next higher layer should respond with a MLME-Associate.response primitive either permitting or denying association. If the *AssociatedAddress* is less than 8, the association will be permitted by issuing an MLME-association response with the *AssociatedAddress* and a status of success. If the AssociatedAddress is greater than 8, association will be denied and a status of PAN at Capacity will be issued. In this case the associated short address parameter is set to the unassociated state of *0xFFFF*.

Note that demo code includes code for both Coordinator and Device. Some mac to network primitives are only used by Coordinator or Device. The *IsCoordinator* variable can be used to differentiate cases that require different behavior for Coordinator and Device.

9. Using the Library

The project file includes special provisions to include the 802.15.4 MAC Library file in the list of files to be linked. The Library file is located in the following directory:

Silabs\MCU\Examples\802_15_4\MAC_Blinky\lib

Three external obj/lib files have been added to the build. To view or alter the files to be linked:

1. Select Target Build Configuration” from the Project menu
2. Click on the “Customize” button
3. Select the “Files to Link Tab”

Any source files added to the project will automatically add the corresponding object file. Any object or library file not associated with a source file must be added manually. In the MAC_Blinky example you should see three additional files added to the build:

HELICOMM_MAC.LIB

hal_int0_isr.obj

hal_pca0_isr.obj

In addition to the library file there are two object files for essential interrupt service routines that have been added to the project. This ensures that all interrupt driven code will be included from the library.

If you would like to add the library file to another project follow these steps:

1. Select Target Build Configuration” from the Project menu
2. Click on the “Customize” button
3. Select the “Files to Link Tab”
4. Click on the Add External OBJ

5. Select “All Files” from the “List files of Type” pull-down menu
6. Browse to the library file location
7. Select the HELICOMM_MAC.lib file
8. Repeat for the two interrupt service routine OBJ files

The process for adding the MAC library to a project is illustrated in Figure 9.

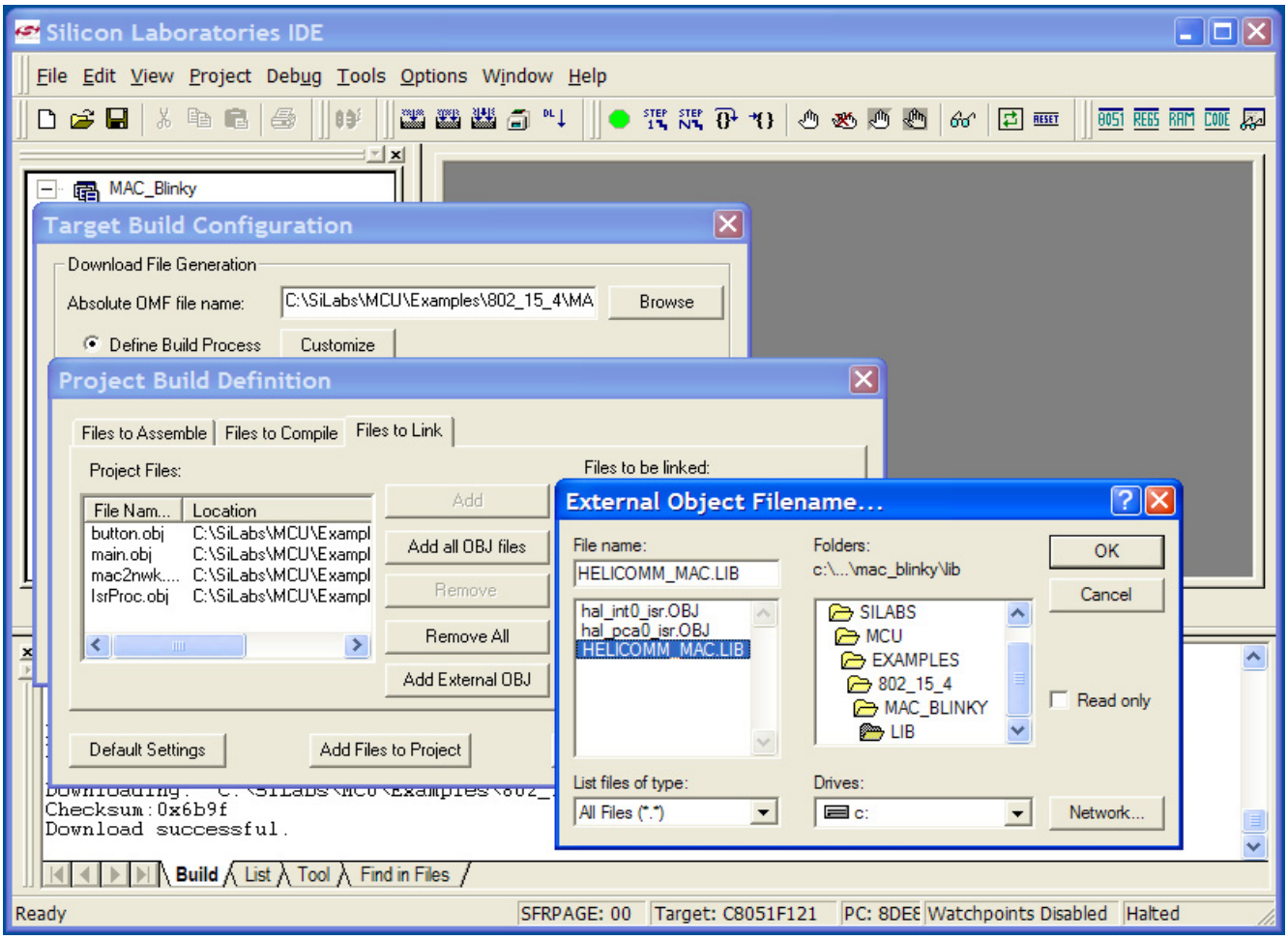

Figure 9. Including an External Library File

10. Library Limitations

- The MAC Blinky example uses most of the available code size for the evaluation tools. It may be impractical to make substantial changes without a full version of the Keil tools.
- Direct Access to the lower MAC and PHY layers is not supported.
- The library only supports the C8051F121.
- The library does not support code banking.
- The library must be used with the Silicon Laboratories version of the BL51 linker version 5.15 or greater.

Source code is not distributed with the 2.4 GHz 802.15.4 Development kit. Source code for the MAC software is available only with a signed license agreement. Contact mcuapps@silabs.com to obtain a source code license.

Refer to the IEEE 802.15.4-2003 MAC specification for a functional description of the 802.15.4 MAC layer and MAC primitives. Refer to “AN268: MAC API Users Guide” for additional information on using the library MAC function calls.

NOTES:

CONTACT INFORMATION

Silicon Laboratories Inc.

4635 Boston Lane
Austin, TX 78735
Email: MCUinfo@silabs.com
Internet: www.silabs.com