ATC 2008

MSP430 Advanced Technical Conference

RF Networking With the MSP430 & eZ430-RF2500 – Session 1

Miguel Morales, MSP430 Applications

Texas Instruments

Agenda

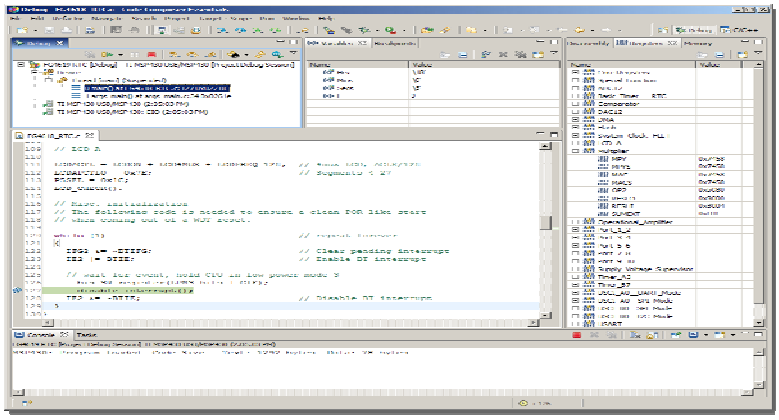
- SimpliciTI Overview
- Lab Hardware Description
- Session 1 Lab Overview
- Lab 1.1 – Wired Sensor Monitor [UART]
- Lab 1.2 – SimpliciTI Semaphores
- Lab 1.3 – Integrating Existing Application
- Lab 1.4 – Enabling Wireless Communication

Hardware Setup

- Connect eZ430-RF2500T targets onto RF header
  - Remember to connect JP3! (RF Vcc)
- Connect MSP-FET430UIF to PC
  - Connect JTAG to EXP5438 JTAG header
  - If you have not installed the driver, please do so at this time
- Connect USB cable to PC COM
  - If you have not installed the driver, please do so at this time
  - Identify serial port → open MS Device Manager
    - Right click 'My Computer' > Properties > Hardware > Device Manager > Ports (COM & LPT)'
    - Look for 'USB – Serial Port (COMxx)'

Software Setup

- If you have not done so, please install CCE



Agenda

- SimpliciTI Overview
- Lab Hardware Description
- Session 1 Lab Overview
- Lab 1.1 – Wired Sensor Monitor [UART]
- Lab 1.2 – SimpliciTI Semaphores
- Lab 1.3 – Integrating Existing Application
- Lab 1.4 – Enabling Wireless Communication

What is SimpliciTI?

- **Low Power**: Supports sleeping devices for low power consumption
- **Low Cost**: uses <8K FLASH and <1K RAM depending on platform
- **Flexible**: simple star w/ extender and/or p2p communication
- **Simple**: Utilizes a very basic core 6 instructions API
- **Versatile**: MSP430+CC1100/2500, CC111x/251x and CC2430/31

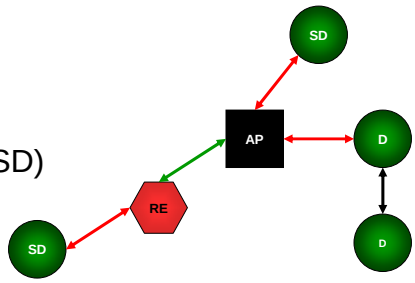
SimpliciTI targets quick time-to-market wireless solutions for low power, low cost, and low data rate networks without the need to know the details of the network support.

What is the Market?

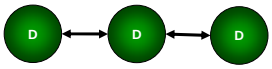
- **Alarm & Security:** Occupancy sensors, light sensors, carbon monoxide sensors, glass-breakage detectors
- **Smoke Detectors**
- **AMR:** Gas meters, water meters, e-meters
- **Home Automation:** Garage door openers, appliances, environmental devices
- **And many more:** Flexible peer-to-peer communication

Networking Topologies

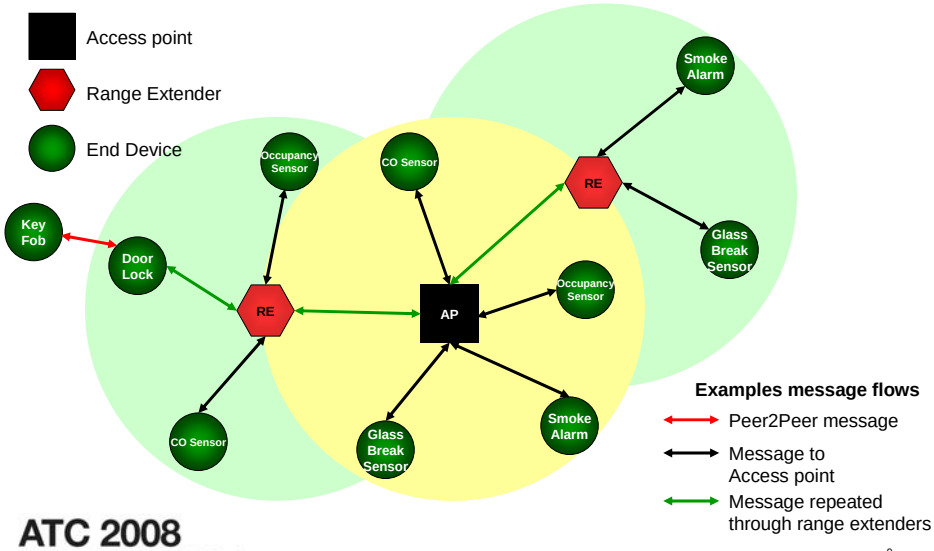
- Device Configurations
  - Access Point (AP)
  - Repeater (RE)
  - Sleeping End Device (SD)
  - End Device (D)



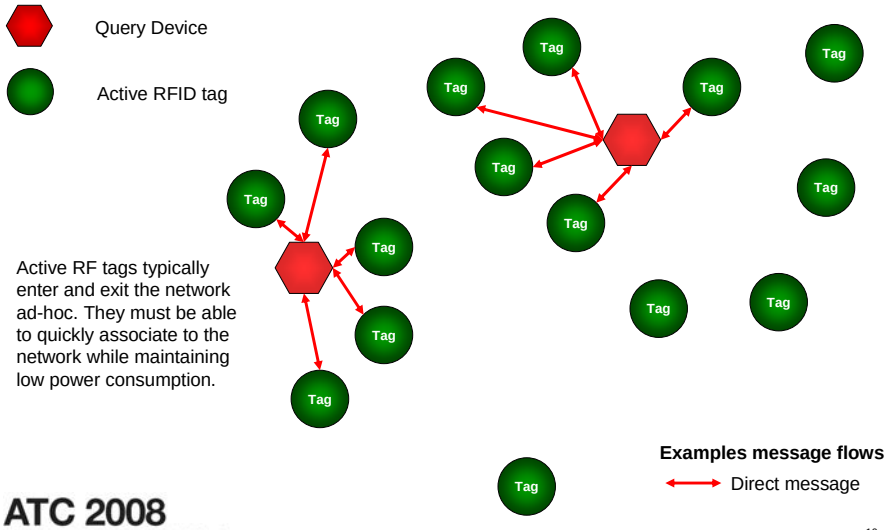
- Topologies
  - AP Star
  - AP Star w/ Repeaters
  - Peer2Peer



Network topologies with SimpliciTI
Wireless Sensing Application

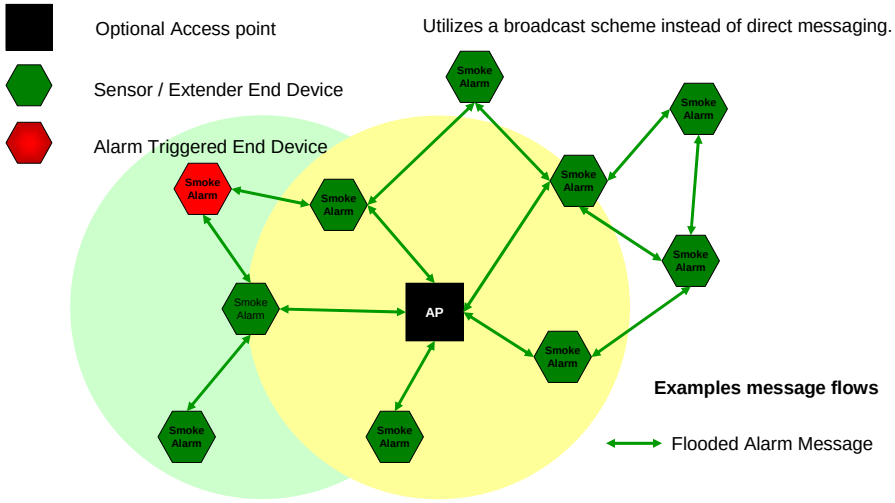


Network topologies with SimpliciTI
Active RF tags

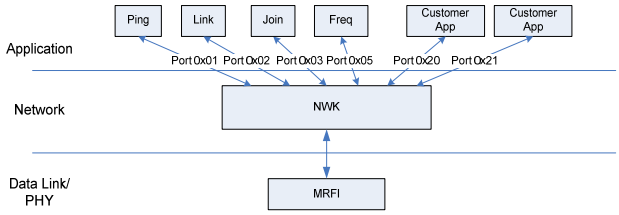


Network topologies with SimpliciTI

Smoke Detector System



Architectural overview



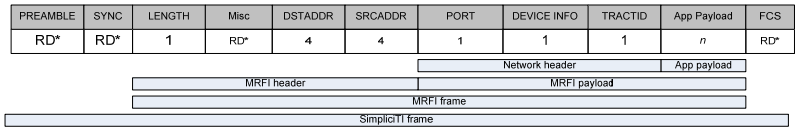
- Layers
  - MRFI
  - NWK
  - nwk applications (Ports < 0x20)
  - customer applications (Ports > 0x20)
- Network Support
  - init
  - ping*
  - link / linklisten
  - nwk mgmt
  - send / receive
  - I/O

* Intended for debug only

Application programming interface (API)

- Initialization
  - smplStatus_t SMPL_Init(uint8_t (*callback)(linkID_t));
- Linking
  - smplStatus_t SMPL_Link(linkID_t *linkID);
  - smplStatus_t SMPL_LinkListen(linkID_t *linkID);
- Peer-to-peer messaging
  - smplStatus_t SMPL_Send(lid, *msg, len);
  - smplStatus_t SMPL_Receive(lid, *msg, *len);
- Configuration
  - smplStatus_t SMPL_Ioctl(object, action, *val);
- API calls are synchronous
  - Do not return until operation is complete

SimpliciTI Frame Structure



*RD: Radio-dependent populated by MRFI or handled by the radio itself

Field	Definition	Comments
PREAMBLE	Radio synchronization	Inserted by Radio HW
SYNC	Radio synchronization	Inserted by Radio HW
LENGTH	Length of remaining frame in bytes	Inserted by FW on Tx, Partially filterable on Rx.
MISC	Radio dependent (needed for future IEEE radio support)	Currently set to 0.
DSTADDR	Destination address	Inserted by FW. LSB filterable. 0x00 and 0xFF LSB values reserved for broadcast. LSB:MSB formatted.
SRCADDR	Source address	Inserted by FW
PORT	Application port number (bits 5-0)	Inserted by FW. Port 0x20-0x3D for customer applications, Port 0x00-0x1F for NWK applications
DEVICE INFO	Receiver type (bit 7-6), Sender Type (5-4) & Hop count (2-0)	Inserted by FW.
TRACTID	Transaction ID	Inserted by FW. Discipline depends on context.
APP PAYLOAD	Application data	$0 \leq n \leq 52$ (50 if FCS)
FCS	Radio append bytes	CRC checksum (Tx), RSSI, LQI and CRC status (Rx)

Build Time Configurations - General

Item	Default Value	Description
MAX_HOPS	3	Maximum number of times a frame is re-sent before the frame is dropped. Each RE and the AP decrement the hop count before re-sending the frame.
MAX_HOPS_FROM_AP	1	Maximum distance an polling ED can be from the AP. To reduce broadcast storm.
NUM_CONNECTIONS	4	Number of links supported. Should be 0 if the device supports no ED objects (APs or REs)
MAX_APP_PAYLOAD	10	Maximum number of bytes in the application payload
SIZE_INFRAME_Q	2	Number of frames held in the RX frame queue.
SIZE_OUTFRAME_Q	2	Number of frames held in the TX frame queue. Some NWK applications keep TX frame around to find correct replies.
DEFAULT_JOIN_TOKEN	0x01020304	Joining a network requires this value to match on all devices (D, SD, RE, and AP).
DEFAULT_LINK_TOKEN	0x05060708	Obtaining a link access to a network device requires this value to match on all devices.
THIS_DEVICE_ADDRESS	0x12345678	Each device address should be unique.

Build Time Configurations – Device Specific

Access Point Devices		
ACCESS_POINT	Defined	
NUM_STORE_AND_FWD_CLIENTS	10	Number of polling End Devices supported.
AP_IS_DATA_HUB	Not Defined	If this macro is defined the AP automatically listens for a link message from each distinct device that joins and supports End Device objects. The ED joining must link immediately after it receives the Join reply.
Range Extender Devices		
RANGE_EXTENDER	Defined	
End Devices		
END_DEVICE	Defined	
RX_POLLS RX_ALWAYS	RX_ALWAYS	Exactly one of these must be defined. This information is included in each frame sent by the device.

- Changing these build-time configurations will affect the volatile memory requirements, which should be kept as low as possible.

Runtime Configuration

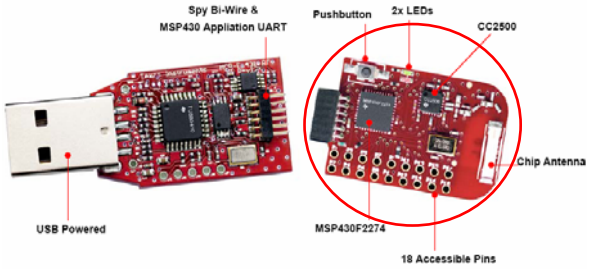
- Application access to frame header
- Application access to radio controls
- Access Point network management control

Object	Description	Comments
IOCTL_OBJ_RAW_IO	Application layer access to the frame header to directly send or receive a frame.	This object is used for example to ping another device where the network address of the target device is supplied directly and not done through the connection table.
IOCTL_OBJ_RADIO	Application layer access to some radio controls.	Limited access to radio directly. For example, sleeping and awakening the radio and getting signal strength information.
IOCTL_OBJ_AP_JOIN	Access Point join-allow context	Interface to control whether Access Point will allow devices to join or not.
IOCTL_OBJ_ADDR	Get / set device address	Permits run-time address generation for a device. Set function must be done before the SMPL_INIT() call

Agenda

- SimpliciTI Overview
- Lab Hardware Description
- Session 1 Lab Overview
- Lab 1.1 – Wired Sensor Monitor [UART]
- Lab 1.2 – SimpliciTI Semaphores
- Lab 1.3 – Integrating Existing Application
- Lab 1.4 – Enabling Wireless Communication

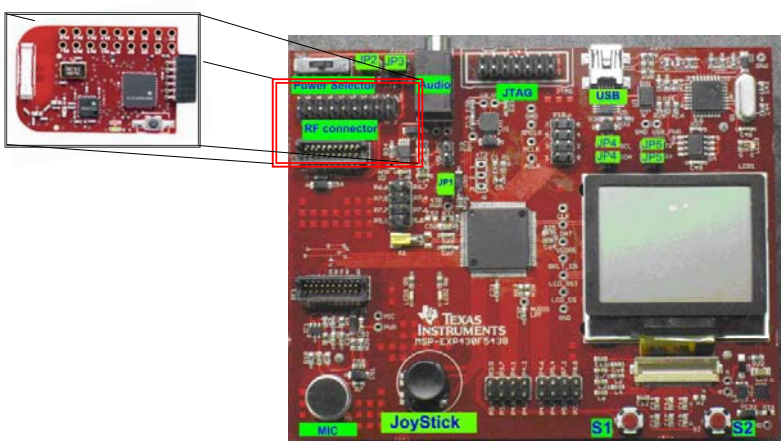
EZ430-RF2500 Development Kit



- **MSP430 → CC2500:**
  - 4 x SPI pins
  - 2 x GPIO pins
  - Made accessible to bypass the x2274

- **Kit includes:**
  - 2 x eZ430-RF2500T target boards
  - 1 x eZ430-RF USB emulator
  - 1 x AAA battery pack w/expansion board
  - Development tools (IAR Kickstart)
  - Wireless Sensor Monitor Demo
  - Documentation (SimpliciTI, Tools)

MSP430EXP5438 Experimenter's Board

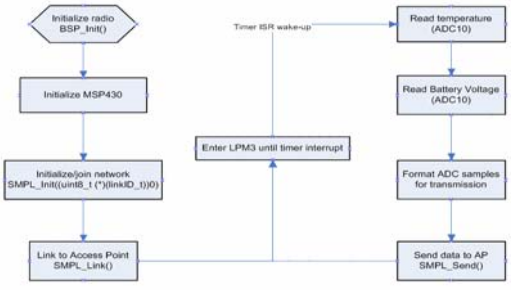


Agenda

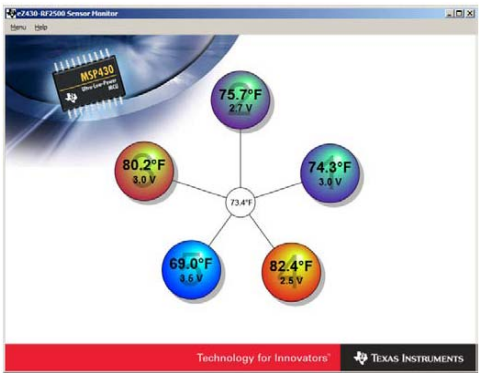
- SimpliciTI Overview
- Lab Hardware Description
- Session 1 Lab Overview
- Lab 1.1 – Wired Sensor Monitor [UART]
- Lab 1.2 – SimpliciTI Semaphores
- Lab 1.3 – Integrating Existing Application
- Lab 1.4 – Enabling Wireless Communication

Session 1 Lab Overview – Lab Goal

- Integrate a wired temperature & vcc sensing application into SimpliciTI project, enabling wireless transmission of application data



Session 1 Lab Overview – Wireless Sensor Demo

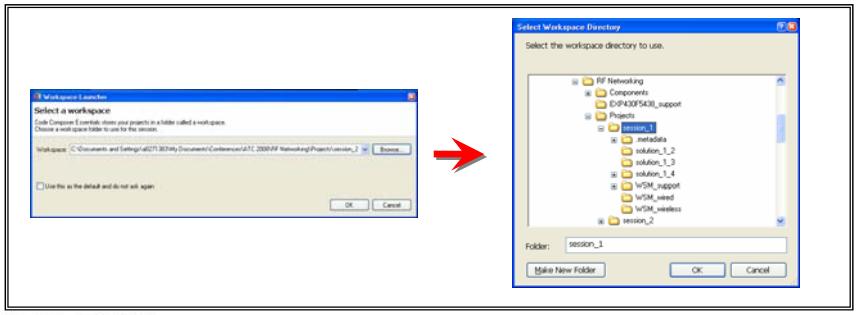


Hardware Setup

- Connect eZ430-RF2500T targets onto RF header
  - Remember to connect JP3! (RF Vcc)
- Connect MSP-FET430UIF to PC
  - Connect JTAG to EXP430F5438 JTAG header
- Connect USB cable to PC
  - Identify serial port → open Device Manager
    - Right click 'My Computer' > Properties > Hardware > Device Manager > Ports (COM & LPT)'
    - Look for 'USB – Serial Port (COMxx)'

Software Setup

- Open CCE
  - Start Menu > All Programs > Texas Instruments > Code Composer Essentials



Agenda

- SimpliciTI Overview
- Lab Hardware Description
- Session 1 Lab Overview
- Lab 1.1 – Wired Sensor Monitor [UART]
- Lab 1.2 – SimpliciTI Semaphores
- Lab 1.3 – Integrating Existing Application
- Lab 1.4 – Enabling Wireless Communication

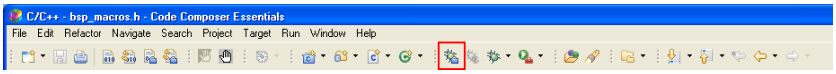
Lab 1.1 Goals

- Learn how to open and debug a CCEv3.0 project
- Learn how to use the Terminal window eclipse plug-in
- Become familiar with the wired temperature and vcc sensor application
  - Will be ported to a SimpliciTI project in Lab 1.2!

Lab 1.1 – Wired Sensor Monitor [UART]

(5 minutes)

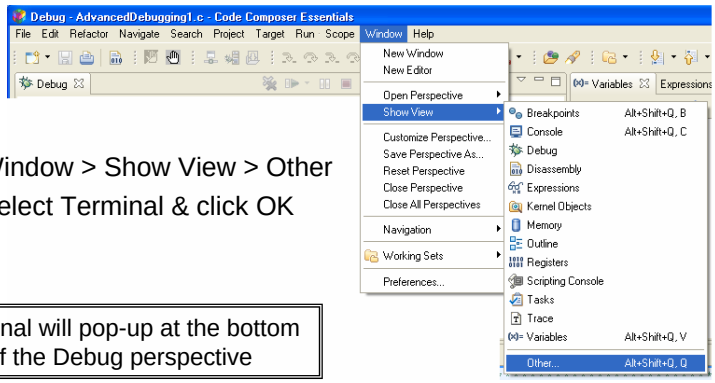
- Verify that the active project is set to WSM_wired
- Run your project by clicking
  - Run > Debug Active Project



- May be prompted that firmware on MSP-FET430UIF is not up-to-date; click Update to continue download.
- Will be prompted that the XMS silicon does not match the expected device; click Ignore to continue download

Lab 1.1 – Wired Sensor Monitor [UART]

- Opening a terminal window

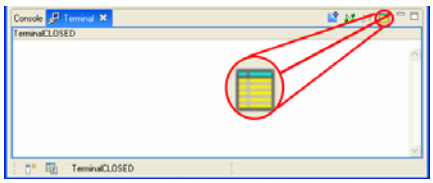


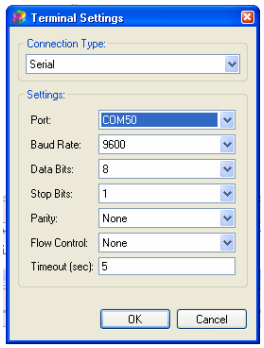
- Window > Show View > Other
- Select Terminal & click OK

Terminal will pop-up at the bottom of the Debug perspective

Lab 1.1 – Wired Sensor Monitor [UART]

- Change the settings
  - The settings button →





Select COMxx from Device Manager	
Baud rate	9600 bps
Data Bits	8
Stop Bits	1
Parity	None
Flow Control	None

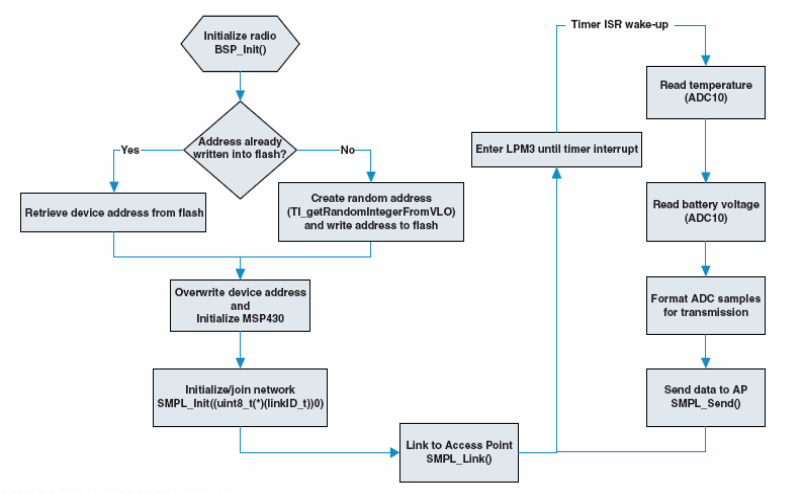
Agenda

- SimpliciTI Overview
- Lab Hardware Description
- Session 1 Lab Overview
- Lab 1.1 – Wired Sensor Monitor [UART]
- Lab 1.2 – SimpliciTI Semaphores
- Lab 1.3 – Integrating Existing Application
- Lab 1.4 – Enabling Wireless Communication

Lab 1.2 Goals

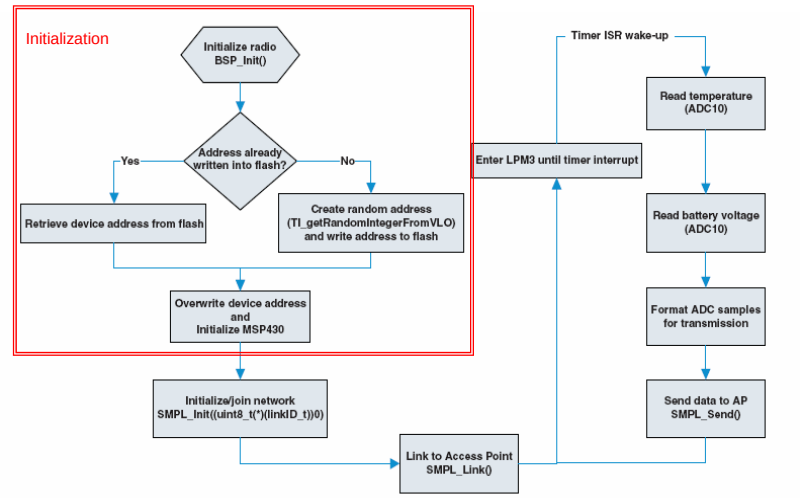
- Introduce the SimpliciTI program flow
  - Semaphores
    - A flag that allows the execution of a certain section of code
- Create a semaphore for the temp and vcc sensor application
- Leverage the MSP430 Timer hardware to improve the power consumption of the application
- Switch the active project to WSM_wireless
  - Right click “WSM_wireless” in C/C++ Projects view
    - Click “Set as Active Project”

Lab 1.2 Intro – SimpliciTI ED Program Flow



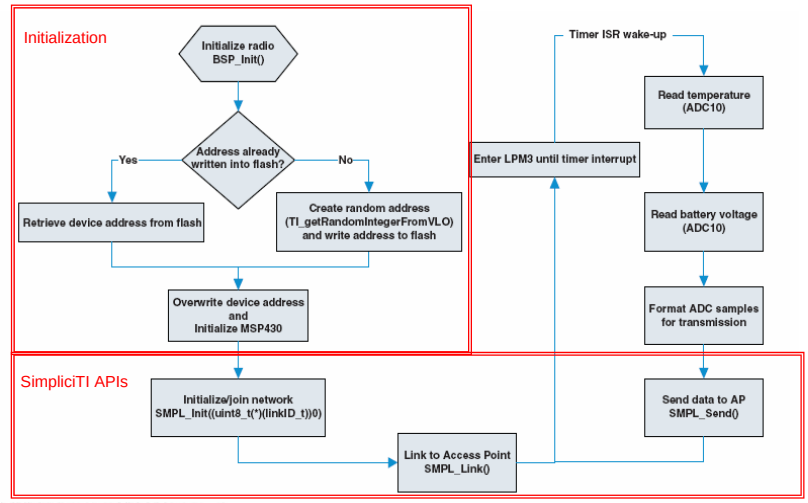
TEXAS INSTRUMENTS

Lab 1.2 Intro – SimpliciTI ED Program Flow

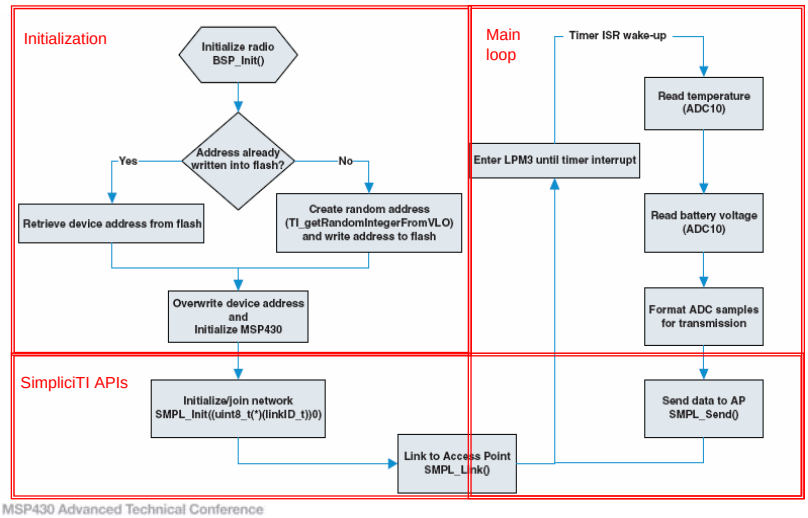


TEXAS INSTRUMENTS

Lab 1.2 Intro – SimpliciTI ED Program Flow

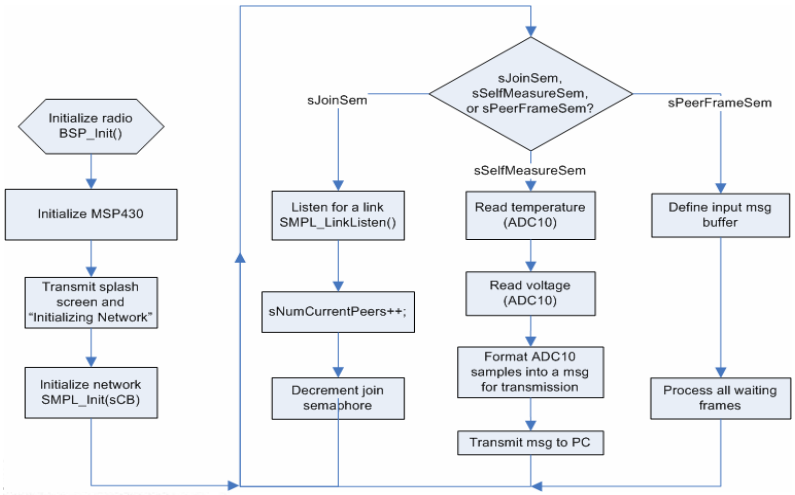


Lab 1.2 Intro – SimpliciTI ED Program Flow



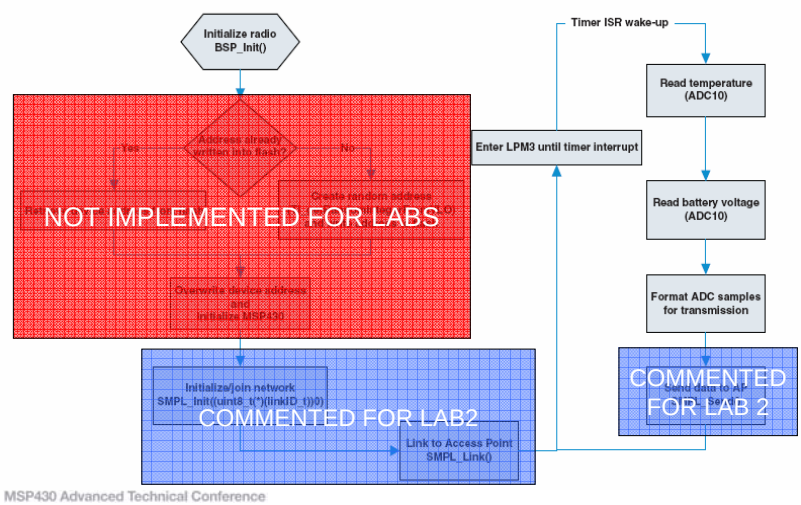
Lab 1.2 Intro – SimpliciTI AP Program Flow

Access Point Implementation



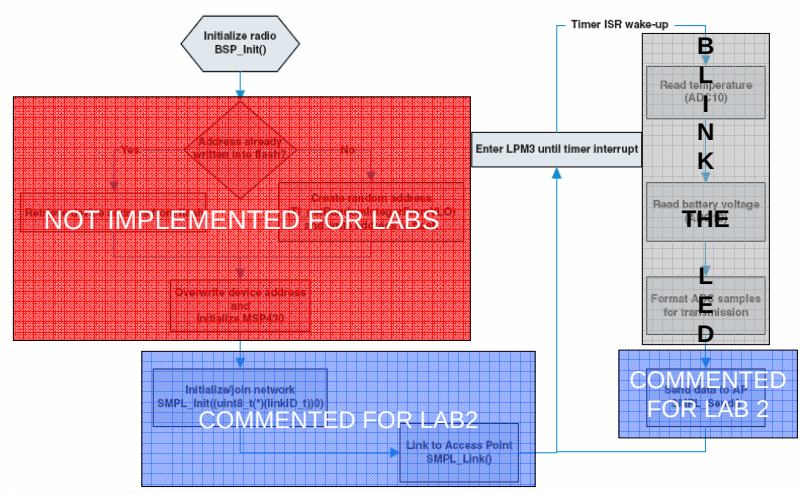
TEXAS INSTRUMENTS

Lab 1.2 Goal – SimpliciTI Semaphores



TEXAS INSTRUMENTS

Lab 1.2 Goal – SimpliciTI Semaphores



Lab 1.2 – SimpliciTI Semaphores

- To do:
  - Find demo_ED.c in the project window; double-click
  - Create a sSelfMeasure semaphore
  - Set sSelfMeasure semaphore inside TimerB ISR

```
#pragma vector=TIMERB0_VECTOR
__interrupt void TimerB_ISR(void)                    {
  set sSelfMeasureSem semaphore;              // = 1
  __bic_SR_register_on_exit(LPM3_bits);   // exit active CPU from ISR }
```

  - Add semaphore check to main loop and uncomment the BSP_TOGGLE_LEDx( ) macros.

```
while( 1 )
{
  Put CPU into LPM3;          // See Tool Box next slide
  if(sSelfMeasureSem){        // Check semaphore
    Toggle LED;               // For debugging...
    sSelfMeasureSem = 0; }
}
```

Lab 1.2 – SimpliciTI Semaphores

(10 minutes)

- Solution can be found in:
  - demo_ED_soln_1_2.c
  - To debug the solution project:
    - Right-click demo_ED.c, select “Exclude File(s) From Build”
    - Right-click demo_ED_soln_1_2.c, un-select “Exclude File(s) From Build”

Lab 1.2 – SimpliciTI Semaphores Solution

- Create a sSelfMeasure semaphore

```
static volatile uint8_t sSelfMeasureSem;          // Now uncommented
```

- Set sSelfMeasure semaphore inside TimerB ISR

```
#pragma vector=TIMERB0_VECTOR
__interrupt void TimerB_ISR(void)
{
   sSelfMeasureSem = 1;

   __bic_SR_register_on_exit(LPM3_bits);          // exit active CPU from ISR
}
```

Lab 1.2 – SimpliciTI Semaphores Solution

– Add semaphore check to main loop and uncomment the BSP_TOGGLE_LEDx( ) macros.

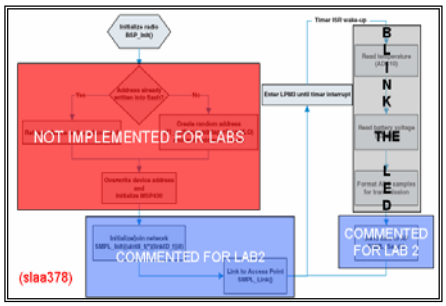
```
while( 1 )
{
   ...
   __bis_SR_register(LPM3_bits);
   ...
   if(sSelfMeasureSem)                  // Check semaphore
   {
      BSP_TOGGLE_LED1( );               // For debugging...
      BSP_TOGGLE_LED2( );
      ...
      sSelfMeaseureSem = 0;             // Reset the semaphore
   }
}
```


Agenda

- SimpliciTI Overview
- Lab Hardware Description
- Session 1 Lab Overview
- Lab 1.1 – Wired Sensor Monitor [UART]
- Lab 1.2 – SimpliciTI Semaphores
- Lab 1.3 – Integrating Existing Application
- Lab 1.4 – Enabling Wireless Communication

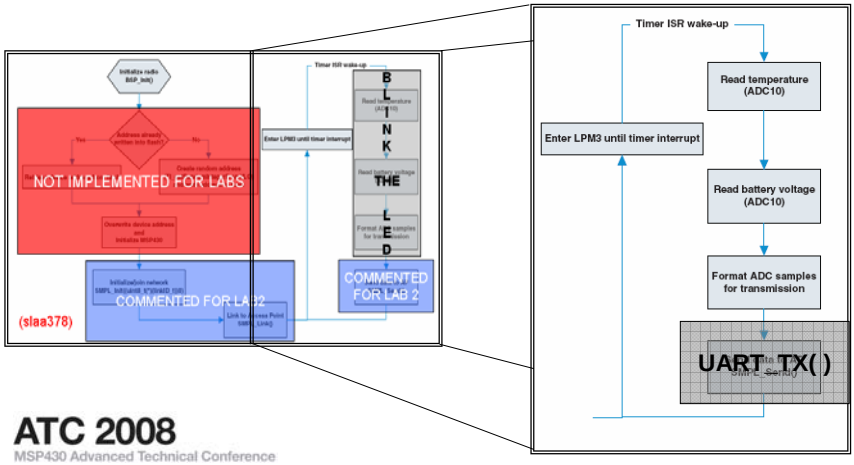
Lab 1.3 Goals

- Lab 1.2 flowchart:



Lab 1.3 Goals

- Lab 1.2 flowchart:
- Lab 1.3 goal:



Lab 1.3 – Integrating Existing Application

- To do:
  - Integrate temperature and Vcc measurement code into demo_ED.c
    - Copy/paste code into demo_ED.c main loop
  - Uncomment the following lines
    - Initial splash and startup transmissions

```
USB_Send_String( (unsigned char *)splash, sizeof(splash));
USB_Send_String( "\r\nInitializing Network....", 26);
...
USB_Send_String( "Done\r\n", 6);
```

  - Comment out or delete the BSP_TOGGLE_LEDx( ) functions!

```
BSP_TOGGLE_LED1( );
BSP_TOGGLE_LED2( );
```

  - Run the project and verify data in Terminal

Lab 1.3 – Integrating Existing Application

(10 minutes)

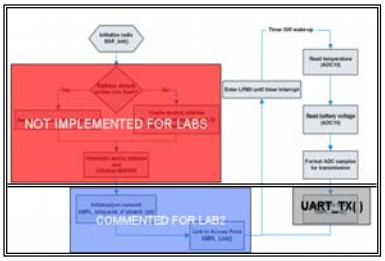
- Solution can be found in:
  - demo_ED_soln_1_3.c
  - To debug the solution project:
    - Right-click demo_ED.c, select "Exclude File(s) From Build"
    - Right-click demo_ED_soln_1_3.c, un-select "Exclude File(s) From Build"

Agenda

- SimpliciTI Overview
- Lab Hardware Description
- Session 1 Lab Overview
- Lab 1.1 – Wired Sensor Monitor [UART]
- Lab 1.2 – SimpliciTI Semaphores
- Lab 1.3 – Integrating Existing Application
- Lab 1.4 – Enabling Wireless Communication

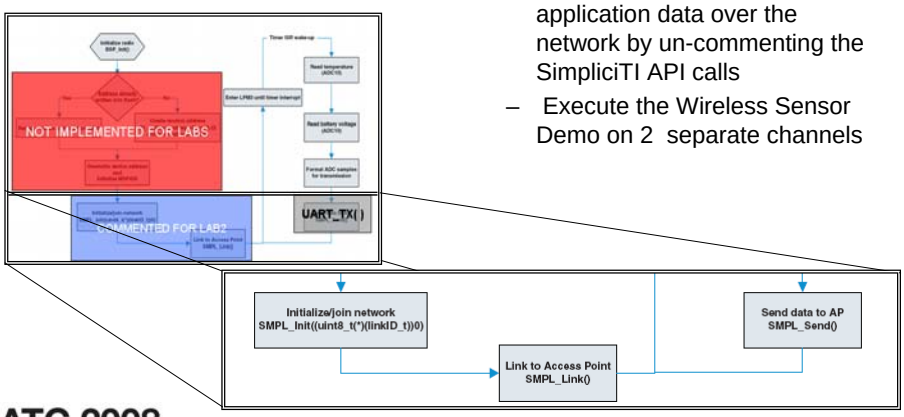
Lab 1.4 Goals

- Lab 1.3 flowchart:



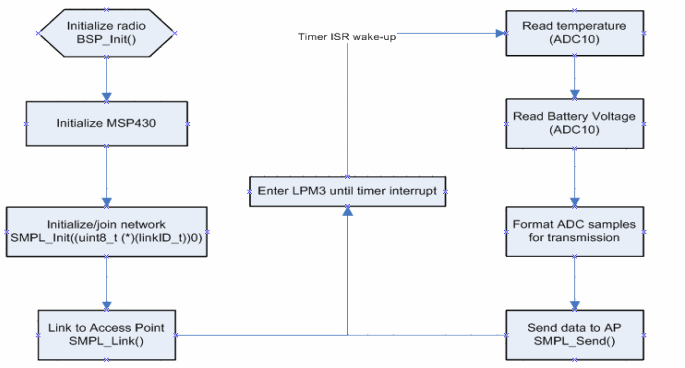
Lab 1.4 Goals

- Lab 1.3 flowchart:



- Lab 1.4 goal:
  - Connect to the AP and send application data over the network by un-commenting the SimpliciTI API calls
  - Execute the Wireless Sensor Demo on 2 separate channels

Lab 1.4 Goals



Lab 1.4 – Enabling Wireless Communication

- To do:
  - Pick a partner!
  - From the paper at your desk:
    - Hard-code the Vcc reading to reflect your personal ID (msg[2] = x)
    - Edit the network address in smpl_config.dat
      - Every network address must be unique

```
// Line 62 in ~/Peer Applications/Configuration/End Device/smpl_config.dat
--define=THIS_DEVICE_ADDRESS="{0xXX, 0x56, 0x34, 0x12}"
```

  - Change the channel of operation in app_support/chann_select.h

```
#define Chan_24XX              // Line xx
```

  - Comment/remove UART TX lines in the code (next slide...)
  - Uncomment all SimpliciTI lines in the code (next slide...)
  - Run and test the code (look to the front of the class!)

Lab 1.4 – Enabling Wireless Communication

- Comment/remove UART TX lines in the code

```
...
USB_Send_String( (unsigned char *) splash, sizeof splash);
USB_Send_String( "\r\nInitializing Network....", 26 );
...
USB_Send_String( "Done\r\n", 6 );
...
transmit_data_string(addr, rssi, msg);
...
```

Lab 1.4 – Enabling Wireless Communication

- Uncommend SimpliciTI lines in code

```
void main (void) {
  ...
  while ( SMPL_SUCCESS != SMPL_Init(...) ){ .... }                    // Init SMPL nwk; join AP
  linkTo( );
}

void linkTo( ){
  ...
  while ( SMPL_SUCCESS != SPML_Link( &sLinkID1 ){ .... }              // Link to the AP
  ...
  while(1){
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, 0 );   // Put radio to sleep
    __bis_SR_register(LPM3_mode + GIE);                         // Sleep CPU until Timer ISR
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_AWAKE, 0 );  // Wake radio into IDLE
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0);     // Radio IDLE → RX
    ...
    if(SMPL_SUCCESS == SMPL_Send(sLinkID1, msg, sizeof(msg)){...};  // send message to AP
    ...
  }
}
```


Lab 1.4 – Enabling Wireless Communication

(10 minutes)

- Solution can be found in:
  - demo_ED_soln_1_4.c
  - To debug the solution project:
    - Right-click demo_ED.c, select “Exclude File(s) From Build”
    - Right-click demo_ED_soln_1_4.c, un-select “Exclude File(s) From Build”

Lab 1.4 – Enabling Wireless Communication

- **SMPL_Init((uint8_t(*)(linkID_t))0)**
  - Initializes the connection to the radio; joins the AP's network
  - A null function pointer → Usually takes a callback function
- **SMPL_Link(&sLinkID1)**
  - Exchanges the Link Token for verification, then creates a bi-directional link between devices. Returns the linkID, or network identifier for the End Device on a network
- **SMPL_Send( )**
  - Sends a packet → Moves a packet to the output buffer
- **SMPL_IOCTL (IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SLEEP, 0)**
  - I/O control API to access radio and frame header parameters; puts radio into SLEEP mode to conserve power

Session 1 Conclusion

- What did we learn?
  - General description of the SimpliciTI protocol
  - How to develop and debug CCE projects, including the terminal plug-in
  - Program flow for SimplicTI End Devices and Access Points in a star topology
  - How to integrate an existing application into the SimpliciTI program flow

Thank you