



NeoMote

PSOC Creator
Component Datasheet
www.metronomesystems.com

Features


System Design

- Ultra-low power consumption
- 50 μ A average (80 μ A with high-precision analog enabled)
- Ultra-low noise (0.5%) 1.25V voltage reference
- Real-time clock with battery backup battery
- Variable power input (3V~25V)
- Multiple power regulator output for a suite of applications
 - 3.3 ~ 5V low-power switching supply (750mA)
 - 3.3 ~ 5V low-noise linear supply (500mA)
- Real-time operating system (RTOS) support
- Full-speed USB interface and mini-USB port
- Over The Air (OTA) updating and reprogramming
- 32 KHz and 20 MHz external crystal sources
- SD-card interface with full file system for local storage

Wireless Sensor Networks

- *Dust Networks'* Eterna™ SoC WSN technology
- 2.4 GHz network operations
- FCC, CE, and IC modular certifications
- Automatic network formation
- Full-mesh networking can easily scale to tens of thousands of nodes
- Time-synchronized communication spanning 15 frequency channels eliminates in-network collisions and multipath fading effects
- Greater than 99.99% network reliability even in the most challenging environments
- Fully engineered RF transceiver, with power amplifier (at +20 dBm)
- Unprecedented low power consumption with an RX current of less than 5 mA and a TX current of less than 10 mA at +8dBm (< 6 mA at 0 dBm)
- AES-128 bit encryption
- Compliant with IETF 6LoWPAN and IEEE 802.15.4e
- IPv6 Internet of Things compliant, enabling each node with a unique Internet-ready IP address

NEOMOTE_1

Neo Mote
Wireless
External Real Time Clock
External Voltage Reference
SD Card




General Description

Metronome Systems provides ultra-low power, highly reliable, true systems-level solutions for a broad suite of real-time sensing and control applications. Our advanced design incorporates a highly-configurable programmable system-on-chip (32-bit ARM® Cortex™-M3 microprocessor unit, memory, full analog and digital peripheral) with industry-leading, ultra reliable, IPv6-based, low-power true-mesh wireless sensor network (WSN) technology. A unique array of configurable analog system blocks features modern methods of signal conditioning, acquisition, and processing to enable monitoring and control with high accuracy, high bandwidth, and high flexibility. The system features a highly configurable digital I/O system with the ability to dynamically configure up to 60 GPIOs with interfaces such as USB, SPI, UART and I2C. Innovative IEEE802.15.4-compliant radio design by *Dust Networks* enables multi-year battery life on a pair of AA batteries. Every system component is rated for industrial applications (-40°C to +80°C), enabling the use of the entire system in extreme environments.

When to Use the NeoMote

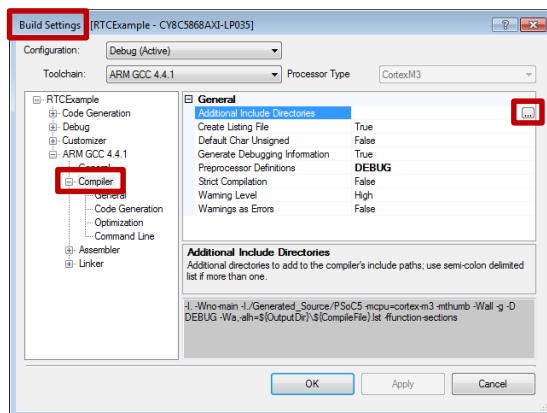
Use the NeoMote component any time an external clock or external voltage reference is needed. The NeoMote provides simple interaction with these peripherals. The use of the NeoMote component is also necessary to access the wireless mesh communication provided by the Dust Networks chip built into the system. Using the NeoMote will allow easy communication with a Dust Networks manager mote through the self-forming network.

Getting Started with the NeoMote



First we have to configure PSoC Creator to know where to look for the libraries associated with this API.

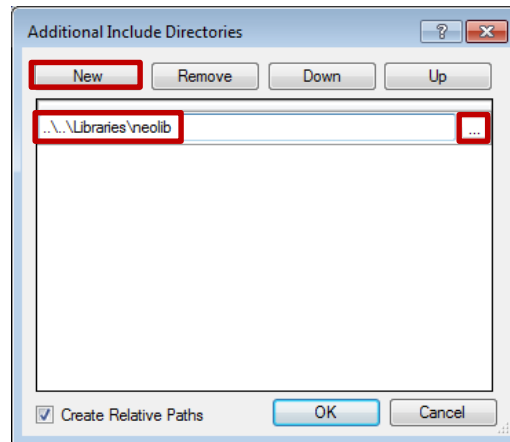
Installing the NeoLib Libraries


- In the Workspace Explorer, right click your project and select **Build Settings**. Under the **ARM GCC 4.4.1**, select **Compiler**.

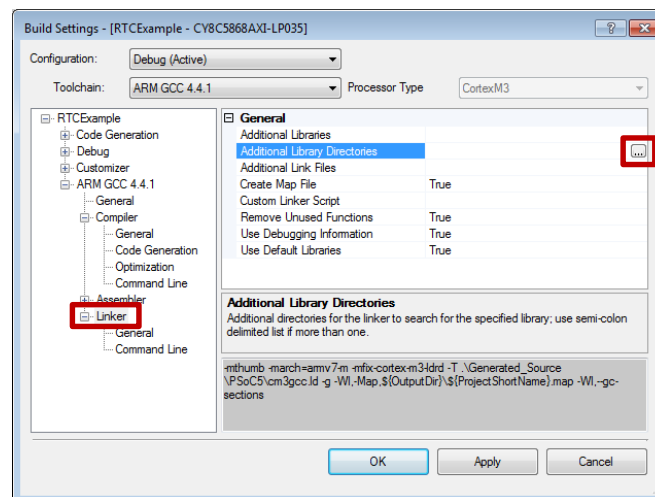




- Click on **Additional Include Directories**. Then click the  button that appears to the right. This will open a new window.
- Click **New**. Then click the  button and navigate to the **neolib** folder that was provided as part of the Metronome files. This tells the compiler where to look for the Metronome Systems libraries.



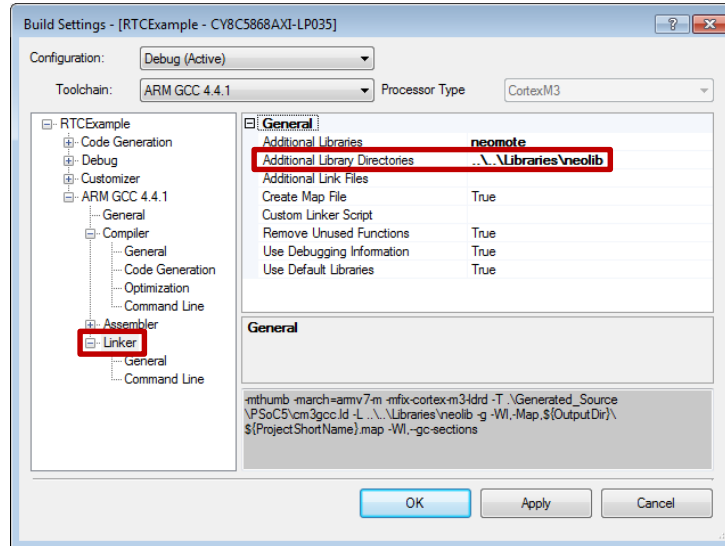
- Click OK.
- Without closing the main setting window, click on the **ARM GCC 4.4.1 -> Linker** item in the same list, and repeat the above process; click **Additional Library Directories**, and use the  button to add the **neolib** library. This tells the linker where to look for the Metronome Systems libraries.



Note: Adding the neolib directory to the linker and compiler adds it to the list of files included when the mote gets programmed. This allows us to use the API's designed by Metronome for the various features of the NeoMote. As development continues, this is where new features can be added.

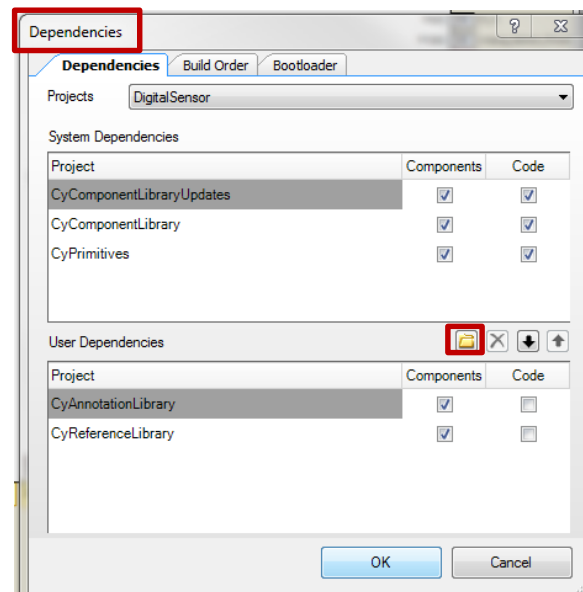


- Under **Additional Libraries** do not browse, but simply type the name **neomote**. This will add the neomote library from the neolib directory.



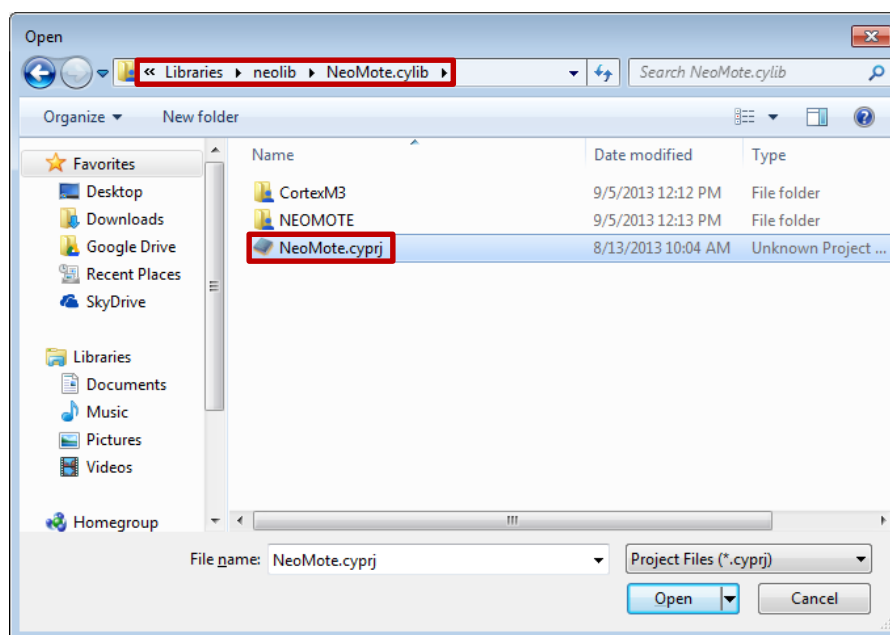
Note: At this point, you would be able to compile your code, but we still need to add the neo library to PSoC creator.

- Right click on your project again, and select **Dependencies**. Under **User Dependencies**, click the  button.





- Navigate to the **neolib** folder given to you by Metronome Systems. Inside this folder, open the **NeoMote.cylib** folder, and then select the **NeoMote.cypri** file.



You should now be able to use the NeoMote component in your schematics.

- Open your Schematic Window by clicking TopDesign.cysch.
- In the component Catalog, you will now see a new tab, called **Default**. Click this tab to reveal the **NEOMOTE** component and drag it to your schematic.

From here, simply assign the pins in the design wide resources file and then read the API description to learn the functions available to you.



Input/output Connections

The following pins must be set up in the .cydwr (Cypress Design Wide Resources) file in order for the NeoMote to function fully.

NeoMote Components	NeoMote Pins
\NEOMOTE_1:External_VRef\	P3[5]
\NEOMOTE_1:I2C_0_SCL\	P12[4]
\NEOMOTE_1:I2C_0_SDA\	P12[5]
\NEOMOTE_1:NEO_RTC_INT1\	P1[6]
\NEOMOTE_1:RX_CTS_n\	P2[2]
\NEOMOTE_1:RX_Pin\	P2[1]
\NEOMOTE_1:RX_RTS_n\	P2[3]
\NEOMOTE_1:SD_Card_Power\	P2[0]
\NEOMOTE_1:TimeN\	P2[7]
\NEOMOTE_1:TX_CTS_n\	P2[5]
\NEOMOTE_1:TX_Pin\	P2[4]
\NEOMOTE_1:TX_RTS_n\	P2[6]

- External_VRef is the connection to the voltage reference.
- I2C_0_SCL is the Serial Clock Line for the I2C connection with the Real Time Clock.
- I2C_0_SDA is the Serial Data Line for the I2C connection with the Real Time Clock.
- NEO_RTC_INT1 is the external line for the Real Time Clock Interrupt.
- RX_CTS_n is the UART Receiver Clear-To-Send line.
- RX_Pin is the UART Receiver Data pin.
- RX_RTS_n is the UART Receiver Request-To-Send line.
- SD_Card_Power is the power line for the external SD card.
- TimeN is a timestamp triggering pin for super accurate timing, independent of OS delay.
- TX_CTS_n is the UART Transmitter Clear-To-Send line.
- TX_Pin is the UART Transmitter Data pin.
- TX_RTS_n is the UART Transmitter Request-To-Send line.



Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name “NeoMote_1” to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is “Mote”.

Function	Description
Mote_Process_Tasks()	Allows the mote to process tasks in the background so the main function can continue.
Mote_Start(void(*callback (NeoMoteNotification notif)));	Initializes and starts mote functions with the appropriate associated callback function.
Mote_Send_Packet(uint8 *packetData, uint8 len);	Sends the specified data packet to the manager.
Mote_Hardware_Busy();	Returns the busy status of the mote.
Mote_Get_State();	Returns the current state of the mote.
Mote_Reset();	Resets the mote.
Mote_Sleep();	Prepares the mote to enter sleep mode.
Mote_Wakeup();	Restores the mote after exiting sleep mode.
NeoRTC_Process_Tasks();	Allows the mote to process tasks in the background so the main function can continue.
NeoRTC_Set_Repeating_Minute_Alarm (uint8 interval);	Sets an interrupt to trigger every “interval” minutes.
NeoRTC_Enable_Second_Interrupt();	Enables an interrupt that triggers each second.
NeoRTC_Disable_Second_Interrupt();	Disables an interrupt that triggers each second.
NeoRTC_Start(void(*RespCommand) (uint8 event));	Initializes and starts the RTC functions with the appropriate associated event response function.
NeoRTC_Sleep();	Prepares the RTC to enter sleep mode.
NeoRTC_Wakeup();	Restores the RTC after exiting sleep mode.

```
void Mote_Start(void(*callBack(NeoMoteNotification notif)));
```

Description: Starts and initializes the mote for wireless communications. Required for other Mote_ functions.

Parameters: void(*callBack(NeoMoteNotification notif)): Requires a NeoMote notification function as a callback.

Return Value: void

Side Effects: None



void Mote_Reset();

Description: Resets the NeoMote.

Parameters: void

Return Value: void

Side Effects: None

void Mote_Sleep();

Description: This is the preferred API to prepare the component for sleep. The Mote_Sleep() API saves the current component state. Call the Mote_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. Refer to the PSoC Creator *System Reference Guide* for more information about power management functions.

Parameters: void

Return Value: void

Side Effects: None

void Mote_Wakeup();

Description: This is the preferred API to restore the component to the state when Mote_Sleep() was called. If the component was enabled before the Mote_Sleep() function was called, the Mote_Wakeup() function will also re-enable the component.

Parameters: void

Return Value: void

Side Effects: Calling the Mote_Wakeup() function without first calling the Mote_Sleep() function may produce unexpected behavior.

uint8 Mote_Send_Packet(uint8 *packetData, uint8 len);

Description: Sends a wireless packet to the manager using the wireless network.

Parameters: uint8 *packetData: Array of data bytes for transmission.
uint8 len: Length of data packet in number of bytes.

Return Value: uint8: 0x01 for Success.

Side Effects: Requires started NeoMote and active network.



uint8 Mote_Hardware_Busy();

Description: Checks the busy status of the mote hardware to avoid usage overlap.

Parameters: void

Return Value: uint8: 0x01 for Busy.

Side Effects: None

uint8 Mote_Get_State();

Description: Retrieves the current state of the mote.

Parameters: void

Return Value: uint8: 0x00 for IDLE.
0x01 for SEARCHING.
0x02 for OPERATIONAL.

Side Effects: None

uint8 NeoRTC_Process_Tasks();

Description: This function allows the user to run RTC tasks such as communications in the background while the rest of the code continues to execute.

Parameters: void

Return Value: uint8: 0x01 for Success.

Side Effects: None

void NeoRTC_Set_Repeating_Minute_Alarm(uint8 interval);

Description: Sets an interrupt to trigger every “interval” minutes to allow the user to process data or perform other tasks.

Parameters: uint8 interval: Duration of alarm in minutes.

Return Value: void

Side Effects: None



`void NeoRTC_Enable_Second_Interrupt();`

Description: Starts an interrupt that triggers every second to allow user to perform various activities on a regular basis.

Parameters: void

Return Value: void

Side Effects: None

`void NeoRTC_Disable_Second_Interrupt();`

Description: Disables the interrupt that triggers every second.

Parameters: void

Return Value: void

Side Effects: None

`void NeoRTC_Start(void(*RespCommand)(uint8 event));`

Description: Starts the NeoMote's external real time clock to provide a settable outside clock source.

Parameters: void(*RespCommand)(uint8 event): Requires event callback function to handle RTC events.

Return Value: void

Side Effects: None

`void NeoRTC_Sleep();`

Description: This is the preferred API to prepare the component for sleep. The NeoRTC_Sleep() API saves the current component state. Call the NeoRTC_Sleep() function before calling the CyPmSleep() or the CyPmHibernate() function. Refer to the PSoC Creator *System Reference Guide* for more information about power management functions.

Parameters: void

Return Value: void

Side Effects: None



```
void NeoRTC_Wakeup();
```

Description: This is the preferred API to restore the component to the state when NeoRTC_Sleep() was called. If the component was enabled before the NeoRTC_Sleep() function was called, the NeoRTC_Wakeup() function will also re-enable the component.

Parameters: void

Return Value: void

Side Effects: Calling the NeoRTC_Wakeup() function without first calling the NeoRTC_Sleep() function may produce unexpected behavior.