

## Wireless Energy Harvesting for the Internet of Things

**Software Used:** NetSim Standard v13.0 (32/64-bit), Visual Studio 2019

### Project Download Link:

[https://github.com/NetSim-TETCOS/Energy\\_Harvesting\\_in\\_IOT\\_v13\\_0/archive/refs/heads/main.zip](https://github.com/NetSim-TETCOS/Energy_Harvesting_in_IOT_v13_0/archive/refs/heads/main.zip)

Follow the instructions specified in the following link to download and setup the Project in NetSim:

<https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects>

### Introduction:

Among different energy harvesting methods, such as vibration, light, and thermal energy extraction, wireless energy harvesting (WEH) has proven to be one of the most promising solutions by virtue of its simplicity, ease of implementation, and availability. This recent technology trend in energy harvesting provides a fundamental method to prolong battery longevity. While harvesting from the environmental sources is dependent on the presence of the corresponding energy source, RF energy harvesting provides key benefits in terms of being wireless, readily available in the form of transmitted energy (TV/radio broadcasters, mobile base stations, and handheld radios), low cost, and small form factor implementation.

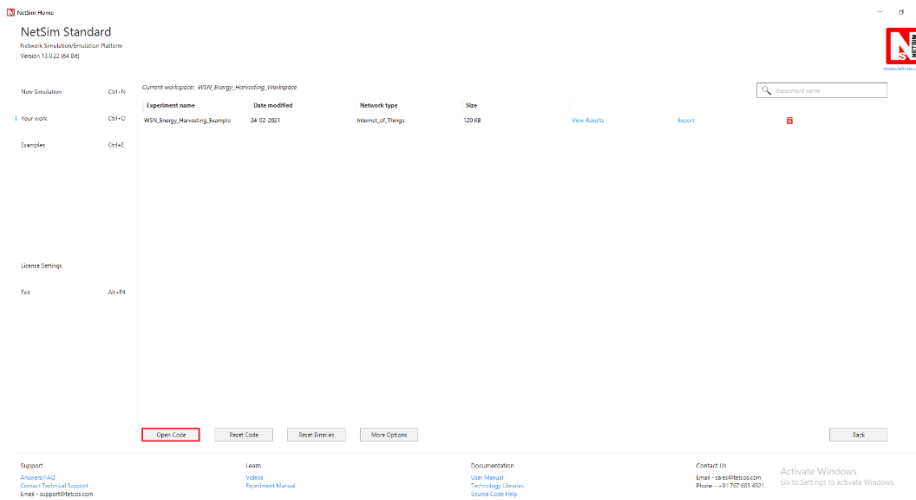
A WEH-enabled sensor device usually consists of an antenna, a transceiver, a WEH unit, a power management unit (PMU), a sensor/processor unit, and possibly an onboard battery. The available harvested power,  $PH$ , is given by a Friis equation and is directly proportional to the transmitted power,  $PT$ , path loss,  $PL$ , transmitter antenna gain,  $GT$ , receiver antenna gain,  $GR$ , power conversion efficiency of the converter,  $PCEH$ , and the square of the wavelength,  $\lambda$ , and is inversely proportional to the square of the communication distance,  $r$ , between the source and the device.

The communication energy consists of  $ELS$  (listening energy),  $ERX$  (receiver energy), and  $ETX$  (transmitter energy). The computation energy includes  $EPR$  (processing energy) and  $ESN$  (sensing energy). To capture the energy distribution among the aforementioned energy consumers, weighting coefficients  $a_{LS} > a_{TX} > a_{RX} > a_{PR} > a_{SN}$  are assigned to them. The total average energy consumption  $ED = a_{LS} ELS + a_{TX} ETX + a_{RX} ERX + a_{PR} EPR + a_{SN} ESN$ .  $EB$  is the total energy stored in the battery, and  $EH$  is the available harvested energy per active duty cycle. We assume constant energy consumptions for receiver, processor, and sensor. However, the energy consumption of the transmitter ( $ETX$ ) is directly proportional to  $r_{ij}^2$ , where  $r_{ij}$  is the distance between the originating device  $j$  and the sink node  $i$  (in ring topology) or the sink node/sensor device (in multihop topology). The harvested energy  $EH$  is inversely proportional to  $r_{ij}^2$  (here  $j$  is the sink node and  $r_{ij} = r_{ji}$ ).

### IEEE Ref Paper:

The code given below is for an example implementation of WEH whereby energy is harvested based on the received signal power. The Steps to be followed for Implementation in NetSim are:

1. Open the Source codes in Visual Studio by going to Your work-> Workspace Options and Clicking on **Open code** button as shown below:



2. Expand BatteryModel Project and double click on the BatteryModel.h file to open it. The following changes(highlighted in red) were done to the code:
 

```
_declspec(dllexport) void battery_animation();
_declspec(dllexport) void battery_metrics(PMETRICSWRITER metricsWriter);
_declspec(dllexport) double battery_get_remaining_energy(ptrBATTERY battery);
_declspec(dllexport) int battery_energy_harvesting(ptrBATTERY battery, double eh_energy);
_declspec(dllexport) double battery_get_consumed_energy(ptrBATTERY battery, int mode);
```

3. Now double click on the BatteryModel.c file to open the file. The following changes(highlighted in red) were done to the code:
 

```
_declspec(dllexport) double battery_get_remaining_energy(ptrBATTERY battery)
{
    return battery->remainingEnergy;
}
```

```
_declspec(dllexport)int battery_energy_harvesting(ptrBATTERY battery, double eh_energy)
{
    double eh_energy_mJ = eh_energy *((pstruEventDetails->dEventTime - battery->modeChangedTime) / 1000000);
    battery->remainingEnergy += eh_energy_mJ;
}
```

- Expand ZigBee project and double click on the ChangeRadioState.c file to open it. At the end of ChangeRadioState.c file the following lines of code are added:

```
#define EH_FRACTION 0.1
// EH_FRACTION is the fraction of the received signal energy that can be
// captured and harvested by the sensor.
int calculate_eh(NETSIM_ID dev1, NETSIM_ID dev2)
{
    double rx_pwr = GET_RX_POWER_mw(dev1, dev2, pstruEventDetails-
>dEventTime);
    double eh_energy= EH_FRACTION *
    rx_pwr; ptrBATTERY battery =
    WSN_PHY(dev2)->battery; if(battery)
    battery_energy_harvesting(battery, eh_energy);
}
```

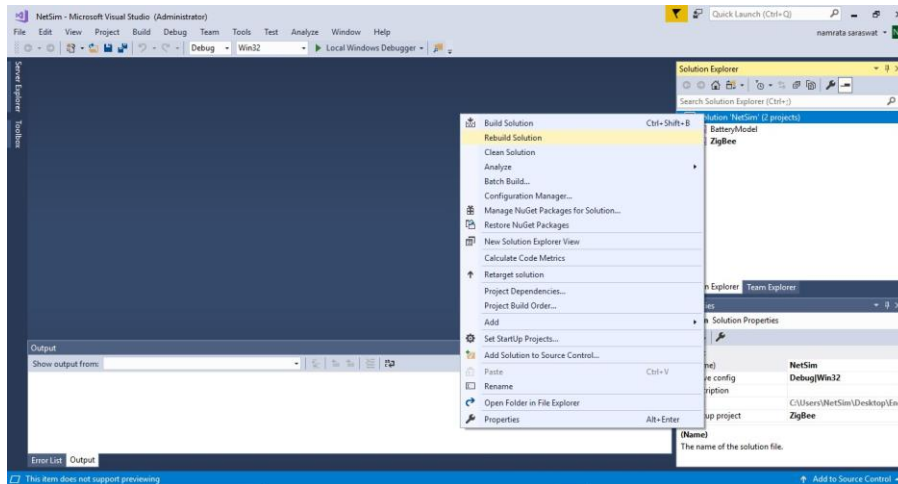
- The function call shown in red is added to 802\_15\_4.c file

```
case UPDATE_MEDIUM:
{
    double dtime=pstruEventDetails->dEventTime;
    NETSIM_ID nLink_Id, nConnectionID, nConnectionPortID,
    nLoop; NETSIM_ID nTransmitterID;
    nTransmitterID = pstruEventDetails->nDeviceId;
    ZIGBEE_CHANGERADIOSTATE(nTransmitterID,
    WSN_PHY(nTransmitterID)-
    >nRadioState, RX_ON_IDLE);
    if(WSN_PHY(nTransmitterID)->nRadioState !=
    RX_OFF) WSN_MAC(nTransmitterID)-
    >nNodeStatus = IDLE;
    nLink_Id = fn_NetSim_Stack_GetConnectedDevice(pstruEventDetails-
    >nDeviceId,pstruEventDetails-
    >nInterfacId,&nConnectionID,&nConnectionPortID);
    for(nLoop=1; nLoop<=NETWORK-
    >ppstruNetSimLinks[nLink_Id-1]-
    >puniDevList.pstruMP2MP.nConnectedDeviceCount; nLoop++)
    {
        NETSIM_ID ncon = NETWORK->ppstruNetSimLinks[nLink_Id-1]-
        >puniDevList.pstruMP2MP.anDevIds[nLoop-
        1]; if(ncon != pstruEventDetails-
        >nDeviceId)
            {
                calculate_eh(nTransmitterID, nLoop);

                WSN_PHY(ncon)->dTotalReceivedPower -=
                GET_RX_POWER_mw(nTransmitterID,ncon,pstruEventDetails-
                >dEventTime);
                if(WSN_PHY(ncon)->dTotalReceivedPower < WSN_PHY(ncon)-
                >dReceiverSensitivity)
                    WSN_PHY(ncon)->dTotalReceivedPower = 0;
            }
    }
}
```

This completes the code modifications for energy harvesting.

- Right click on the Solution in the solution explorer and select Rebuild.

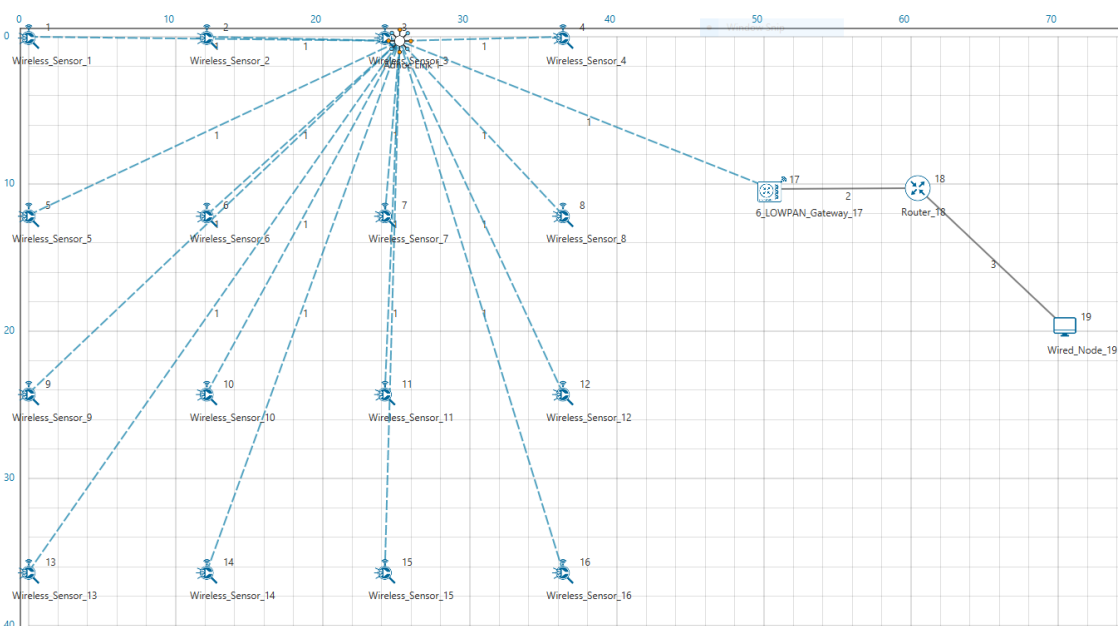


- Upon successful build modified libZigBee.dll and BatteryModel.dll file gets automatically updated in the directory containing NetSim binaries.

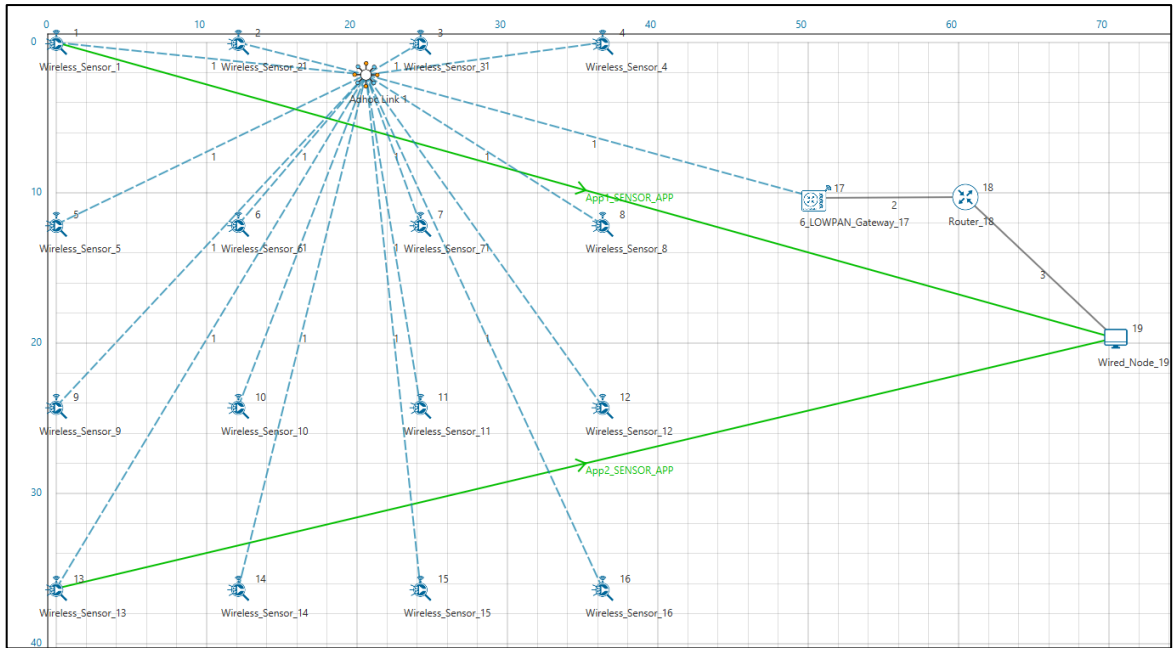
## COMPARATIVE ANALYSIS:

- Create a network scenario in IoT with say 16 sensors, a 6LoWPAN Gateway, a Router and a Wired Node as shown below. The WSN\_Energy\_Harvesting\_Workspace also comes with a sample configuration that is already saved. To open this example, go to Your Work and click on the WSN\_Energy\_Harvesting\_Example that is present under the list of experiments.

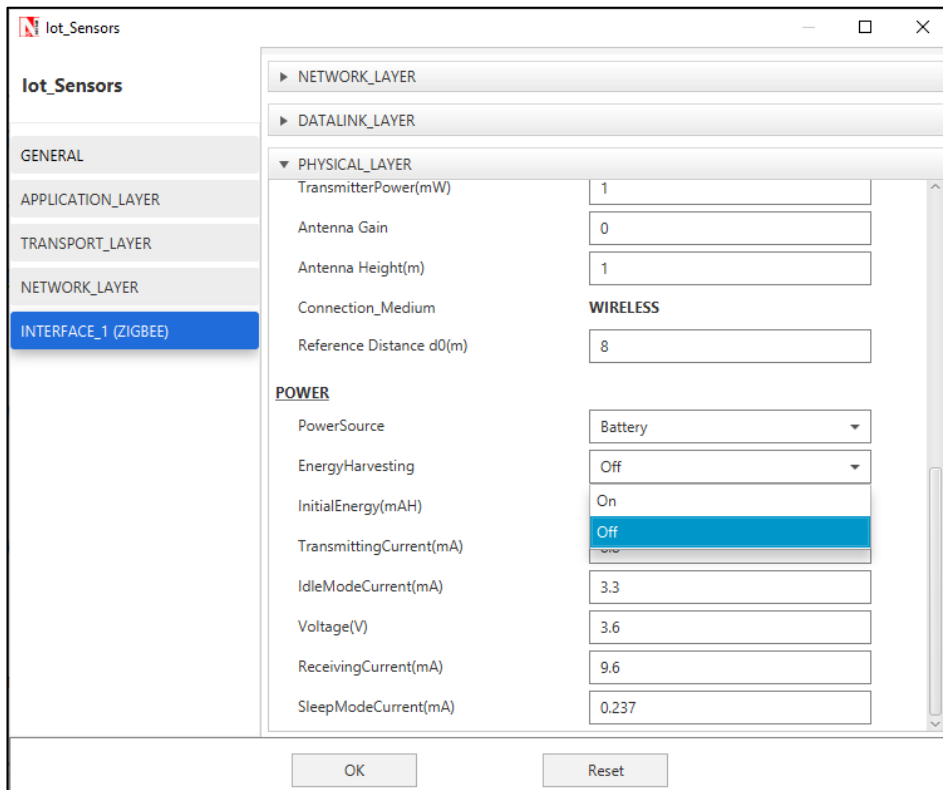
1.



2. Configure traffic in the network by setting a few applications between some of the sensor nodes to the Wired Node using the Application Icon, as shown below:



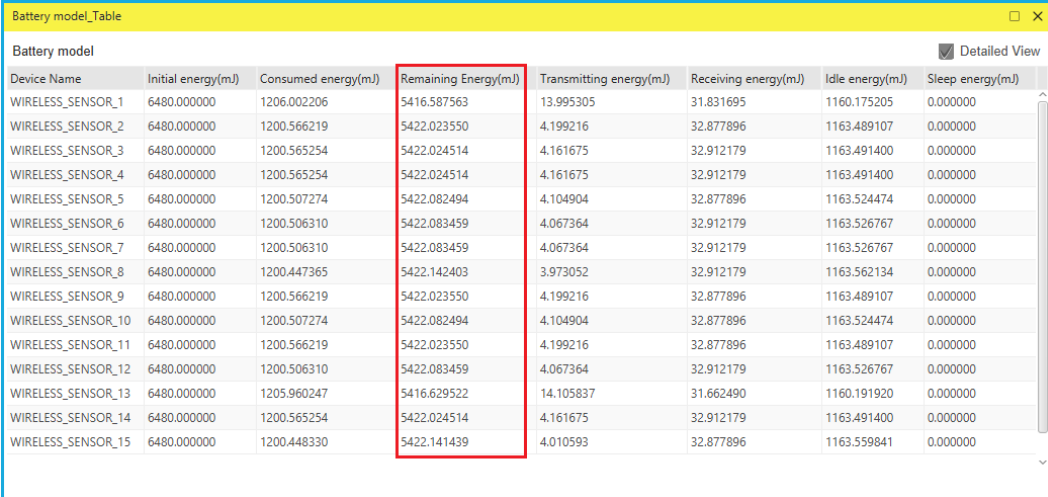
3. Disable Energy Harvesting in all Sensor nodes by setting the EnergyHarvesting parameter to OFF in the Interface(ZigBee) Properties of the sensor nodes as shown below:



4. Run Simulation for say 100 seconds and save the simulation results. In NetSim Simulation Results Window, the Battery model table provides detailed metrics related to energy consumed by each sensor node. The column Remaining energy can be used to compare simulations with and without energy harvesting code modification.

## WITH ENERGY HARVESTING

1. Now re-run the network simulation for 100 seconds and save the simulation results. You can use the table remaining energy column present in the Battery model table which is part of NetSim Simulation Results window to compare simulations with and without wireless energy harvesting.



Device Name	Initial energy(mJ)	Consumed energy(mJ)	Remaining Energy(mJ)	Transmitting energy(mJ)	Receiving energy(mJ)	Idle energy(mJ)	Sleep energy(mJ)
WIRELESS_SENSOR_1	6480.000000	1206.002206	5416.587563	13.995305	31.831695	1160.175205	0.000000
WIRELESS_SENSOR_2	6480.000000	1200.566219	5422.023550	4.199216	32.877896	1163.489107	0.000000
WIRELESS_SENSOR_3	6480.000000	1200.565254	5422.024514	4.161675	32.912179	1163.491400	0.000000
WIRELESS_SENSOR_4	6480.000000	1200.565254	5422.024514	4.161675	32.912179	1163.491400	0.000000
WIRELESS_SENSOR_5	6480.000000	1200.507274	5422.082494	4.104904	32.877896	1163.524474	0.000000
WIRELESS_SENSOR_6	6480.000000	1200.506310	5422.083459	4.067364	32.912179	1163.526767	0.000000
WIRELESS_SENSOR_7	6480.000000	1200.506310	5422.083459	4.067364	32.912179	1163.526767	0.000000
WIRELESS_SENSOR_8	6480.000000	1200.447365	5422.142403	3.973052	32.912179	1163.562134	0.000000
WIRELESS_SENSOR_9	6480.000000	1200.566219	5422.023550	4.199216	32.877896	1163.489107	0.000000
WIRELESS_SENSOR_10	6480.000000	1200.507274	5422.082494	4.104904	32.877896	1163.524474	0.000000
WIRELESS_SENSOR_11	6480.000000	1200.566219	5422.023550	4.199216	32.877896	1163.489107	0.000000
WIRELESS_SENSOR_12	6480.000000	1200.506310	5422.083459	4.067364	32.912179	1163.526767	0.000000
WIRELESS_SENSOR_13	6480.000000	1205.960247	5416.629522	14.105837	31.662490	1160.191920	0.000000
WIRELESS_SENSOR_14	6480.000000	1200.565254	5422.024514	4.161675	32.912179	1163.491400	0.000000
WIRELESS_SENSOR_15	6480.000000	1200.448330	5422.141439	4.010593	32.877896	1163.559841	0.000000

Now on comparing the custom IOT metrics we can observe that Energy Harvesting increases sensors' working capability. Simulations can be performed for different values of EH Fraction which may vary as per the efficiency of the Energy Harvesting unit.