# A Simple Model for CPU Power Consumption in Sensor Networks

**Brenno B. Coelho**

**Justin M. Fiore**

**07/2005**

# 1. Introduction

Wireless sensor networks promise great advantages when compared to wired ones. A wireless sensor network can be composed of hundreds or thousands of nodes. These networks can be used in a large number of different applications, such as: medical applications, climate monitoring, seismic activity monitoring, and so on. Sensor wireless networks have as advantage the fact that the deployment is easier than in a wired network.

Wireless sensor are often used in remote data acquisition, and because of the environment and the number of nodes, it is not desirable that the network require constant maintenance . The cost of replacing one or various nodes, where it is possible to do, can be considerable and must be avoided. For this reason the power consumption is an issue in wireless sensor networks. Increasing the network efficiency of the network can maximize the lifespan after deployment. Another important aspect is that batteries often constitute more than 50% of the sensors weight and volume. For instance, the sensor MICA2 has a weight of 18g and its two AA batteries weigh between 20 and 30 g each [1].

The considerable interest in wireless sensor networks led to the creation of a working group within the IEEE standardization committee, which specified a new standard IEEE 802.15.4. By specifying very low power consumption medium access control (MAC) and physical layers, and thanks to the low complexity protocol and low data rate, this standard makes wireless sensor networks possible [2].

For developers it is crucial to evaluate the energy consumption in order to choose the best algorithm and programming style. In our project we will present an energy consumption model based on the probability of use of sensor's devices (e.g. transmitter, receiver, processor and so on) and on device interruptions.

## 2. The Sensor To Be Modeled

Our model was developed based on the MICA 2 structure but it can be used with other processors with few adjustments (such as power consumption and interruption requirements). The MICA2 uses an Atmel Atmega 128L processor. This processor consumes 8 mA when used in the active mode. When using the TinyOS (TOS) operating system, the processor can be configured to run a sensor application and the network/radio communications stack simultaneously. In our model, we do not consider the use of parallel activities, so, according to time-slots, our processor may either be in active mode (acquiring data, transmitting, receiving or processing) or in sleep mode. This decision is made based on probability (client input) and/or interruption signal coming from the devices (sensor, receiver). The 128L processor must be either on the active mode or on the idle mode to be able to receive the interruption signal.

According to [3], the 128L needs 8 mA when working in the active mode and 3.2 mA when working in the idle mode. Table 1 shows some values obtained for 128L

| Mode | Current |
|---|---|
| Active | 8.0 mA |
| Idle | 3.2 mA |
| ADC Noise Reduce | 1.0 mA |
| Power-Down | 103 uA |
| Power-Save | 110 uA |
| Standby | 216 uA |
| Extended Standby | 223 uA |
| Internal Oscillator | 0.93 mA |

**Table 1: power consumption**

Those values were obtained using a 3 volt power supply.

Any of the modes above can be simulated in the model that we designed. It is only necessary to change the power consumption values in the input section of the code. Figure 1 shows the CPU's Simulink implementation.

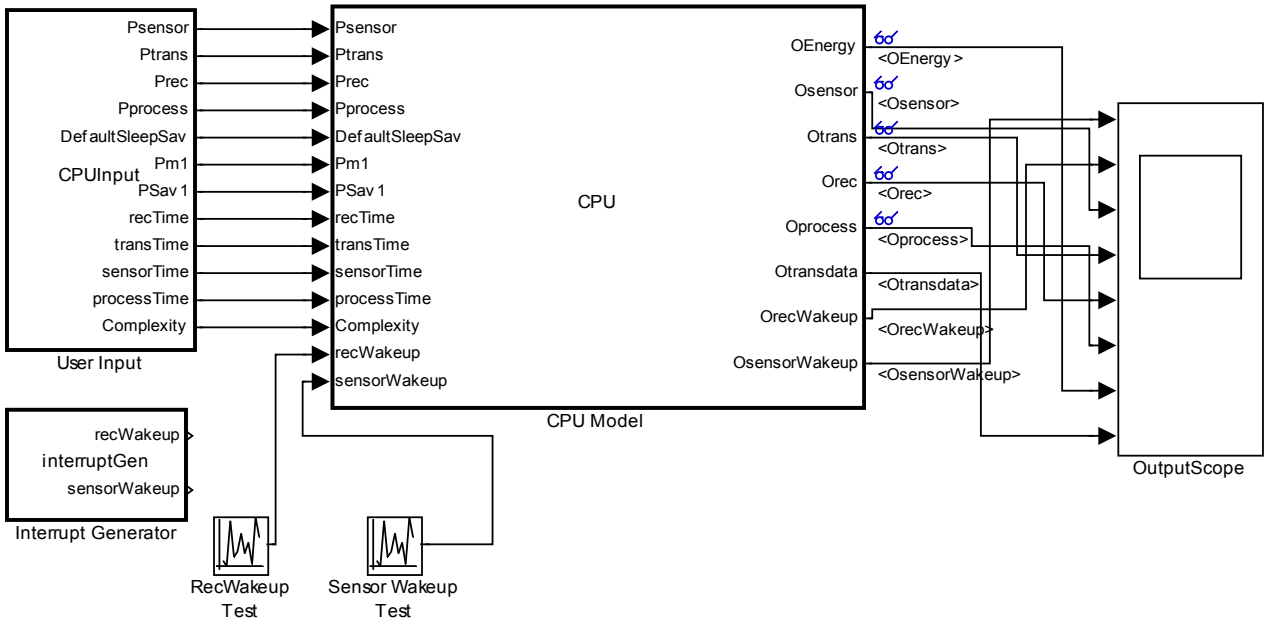


**Figure 1 :Simulink model**

## 3. The CPU Power Consumption Model

Any model for determining the consumption of power of a given CPU tends to be very specific to that CPU. Among the various inputs, are the different power consumption levels for the processors combined with how long the processor is operating in each power level. Because there are countless processors in the current industry, providing an accurate model that is tuned to each of these is impractical. Also, we could have made a model that would take the very specific inputs, such as the power levels and amount of time in each power level, and attempt to accurately compute the power consumption.

Neither of these seemed entirely practical. With the first idea, it is rather obvious that creating a tuned model for each specific processor would be almost impossible. The more direct reasoning behind this is that every processor cannot be simplified into power levels and time in power levels. Also, how is one to find these different values? This can only by doing actual testing on the specific processor. This relates to the second possibility. We could allow specific inputs, but the user of the model would have to measure the specific processor in order to find these inputs. This may not be a problem for some users, but the idea for this model was to create a more general model, that would be usable by a larger set of users. The model is not designed for exact results, but for general results based on more general input.

Now that we know what our model is not, we can describe what the model is. The CPU operation was simplified into cyclic representation. A cycle here is not a CPU cycle, but a cycle of operation. Within each cycle, there are four operations or "segments" that are time division multiplexed: recording sensor data, transmitting data, receiving data, and processing data (see Fig. 2). Each segment has a finite time in which to do its work. Any work not completed, is carried over to the next segment of its type.

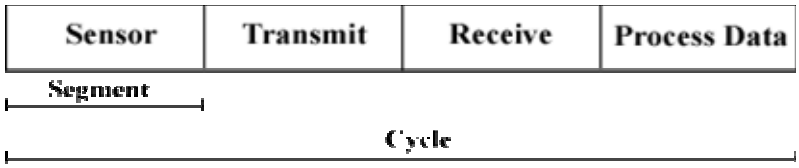


**Figure 2: Simplified CPU Operation**

In each segment, the working entity (transmitter, receiver, etc.) may or may not have work to do. In our model, whether the working entity and the CPU should be awake and running is dependent upon two factors: probability and interrupts. Each working entity has a probability associated with it. During the beginning of the segment, this probability is evaluated and if the working entity does not need to do work, it and the CPU are switched off. By switching it off, we mean that it enters a power

saving mode. In addition, the receiver and the sensor can interrupt the CPU such that if the interrupt is enabled for the sensor during the sensor segment, the sensor and CPU will wake up independent of the sensor probability. This holds true for the receiver interrupt also. The interrupt may not interrupt another segment; it is only evaluated in the appropriate segment.

The full power consumption of the processor is calculated using a scaled calibration set. Essentially, the user enters a Complexity value, a real number between 0 and 4. This is an estimate of the processor's power consumption under load. The user simply estimates their processor's Complexity by comparing it to those processors in the calibration set (Table 2). For example, an i486 processor would have a Complexity of about 2.4. Also, if the user knows the power consumption at load, they can estimate the Complexity based upon that.

The model also allows for two power saving modes, though it could be extended to add more. There is a default power saving mode, which is the mode that the CPU enters if there is no second power saving mode, or the probability dictates that it should not use the second mode. There is also a second power mode. The CPU enters this second mode if it is determined that it should sleep and if the probability of entering this power saving mode is satisfied. Both the default and the custom power saving modes calculate power consumption based upon an input parameter which specifies the percentage of the full power the particular mode consumes.

| ***Complexity*** | ***Power Dissipated in Watts at Load*** | ***Description*** |
|---|---|---|
| 0 | 0 | Lower Bound of Model |
| 1 | 24e-3 | ATmega128L |
| 2 | 1.1 | i386SX-16 |
| 3 | 15.5 | P54C-200 (Pentium 1 @ 200MHz) |
| 4 | 115 | P4-670 (3.8GHz, 2MB L2 Cache) |

**Table 2: Model Calibration Set**

The output of the model is the energy in Joules consumed during the given time segment and also the control signals to turn on the other working entities. If the transmitter is turned on a stream of ones is also output on a data line. If the transmitter is turned off, zeros are transmitted on this line.

## 4. Using Our CPU Consumption Model

Using our model is fairly straightforward, which was one of the major goals in designing this model. The steps for running the model are as follows:

1. Open CPU.mdl in Simulink
2. Enter (double click) the User Input module
3. Edit the input parameters according to the comments (and the description that will follow)
4. Edit the Simulink time stepping parameters in Simulation->Options (set to fixed-length for best results)
5. Save the model
6. Run the model

The output will be in the form of a scope readout. The user may have to right click in the OEnergy graph and click “autoscale” in order to see anything useful.

When the model was in Matlab as it was originally written, it allowed for different segment lengths for each working entity. Integrating it into Simulink required that we not manage the time in Matlab code, and let Simulink manage the time. Therefore, different segment lengths are no longer supported. The best way to run the model is to use fixed-step timing in the Simulink parameters with the desired segment time as the step time. Be sure to set the segmentTime for each working entity to this value also.

The example user input file in Figure 3 will be explained in the following section. Psensor is the probability that the sensor will be awake and operating during the sensor segment. In this example, the sensor is always on during its segment. It will be turned on every cycle. Ptrans, Prec, and Pprocess are similar in their use. Their values should range from 0 to 100. 0 meaning they are never on, and 100 meaning they are always on during their segment. DefaultSleepSav is the percentage of the full power that is used during the default power saving mode. This value can range from 0 to 100 also. A zero for this value means that the power consumed in this power saving mode is 0 Joules.

Pm1 is the probability that when the CPU has nothing to do, it will enter the custom power saving mode. A zero for this value disables the custom power saving mode. 100 will mean that it will always use this power saving mode when the CPU can save power. Psav1 is the same as DefaultSleepSav except for the custom power saving mode. The values of recTime, transTime, sensorTime, and processTime are the time in ms for each of the segments. These should all be the same when used in the Simulink model. Complexity is the value that aids in calculating the load power for the model. This can be a real number between 0 and 4. A zero means that the load power is zero Watts, a highly unlikely scenario, but a lower bound had to be chosen for scaling purposes. The choosing of a Complexity value was discussed in the previous section.

```
%User Input
Psensor = 100;          %Probability of sensor awake (0-100)
Ptrans = 50;            %Probability of transmitter awake
Prec = 50;              %Probability of receiver awake
Pprocess = 20;          %Probability of processing routines awake
DefaultSleepSav = 50; %Power saving for mode 1, percentage of power used compared to full power (0-100)
Pm1 = 50;               %Probability for power mode 1 (0-100)
PSav1 = 25;             %Power saving for mode 1, percentage of power used compared to full power (0-100)
recTime = 10;           %time in ms for the receiving segment
transTime = 10;         %time in ms for the transmitting segment
sensorTime = 10;        %time in ms for the sensoring segment
processTime = 10;       %time in ms for the processing segment
Complexity = 1;         %Choose 0-4 (real numbers are allowed)
```

**Figure 3: Example User Input File**

## 5. Implementation of the Model

Most of the code for this model was written in Matlab. A wrapper function was also created in order to integrate it with Simulink. The interrupts for the received and sensor are generated using Uniform Random Generator blocks in Simulink. The User Input block and the CPU block are embedded Matlab functions. The output is a multi-signal oscilloscope for the outputs; the interrupt inputs are also shown. All the pertinent code is listed in Appendix A and is also included in original Matlab/Simulink form.

It took a relatively long time to integrate the Matlab model into Simulink. During this process, the model structure changed considerably, including the time stepping. However, the core design behind the model did not change. The only feature of the original Matlab model that did not pass through the Simulink integration was the different segment times for different working entities. This model can be extended to support more than two total power saving modes with relative ease. Also, the calibration set can also be changed if the user so desires.

## 6. Results

We simulated for two different periods. First we simulate for t=5 seconds and then for t=10 seconds. Figures 4 and 5 show the results.

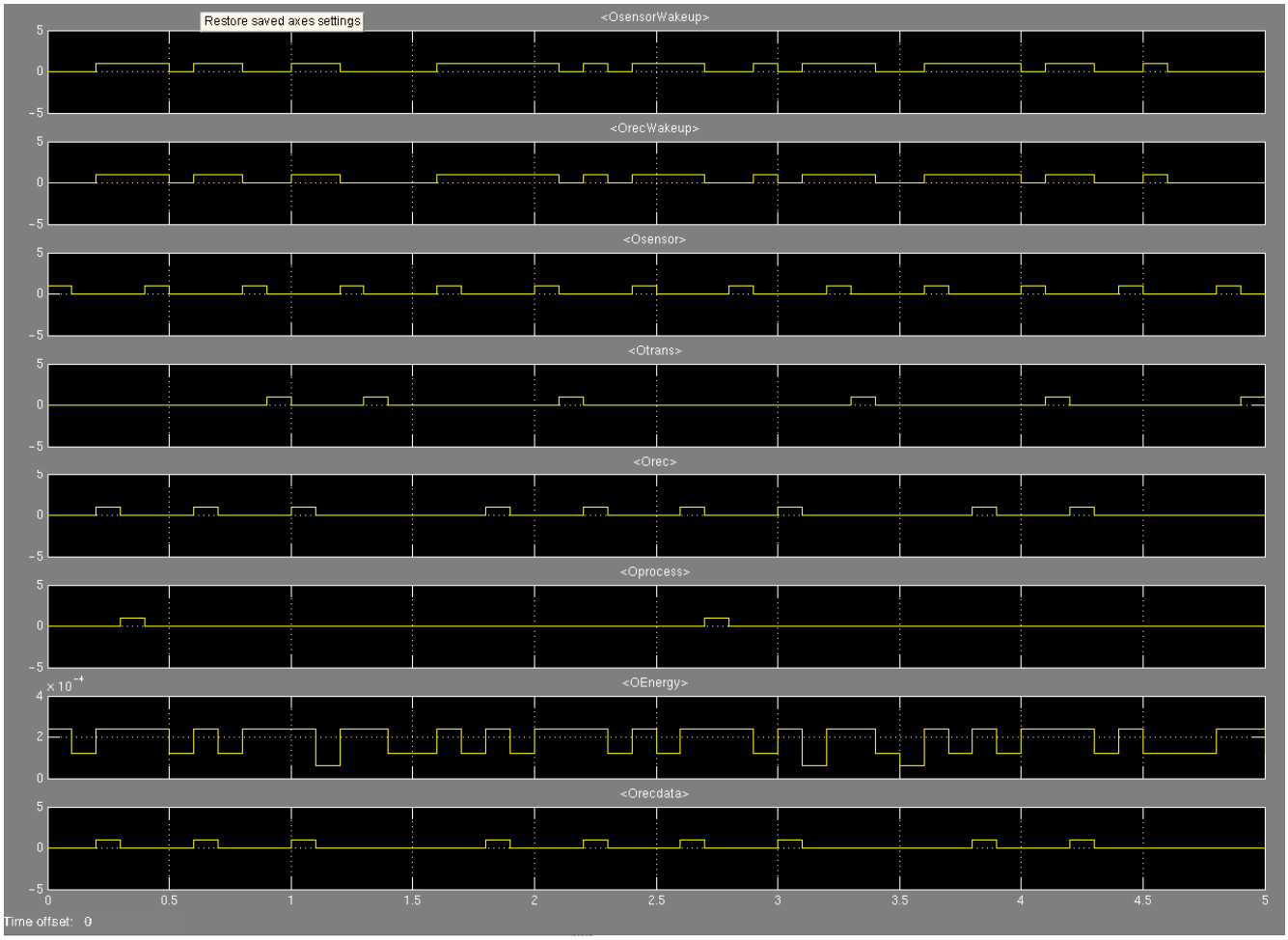


**Figure 4: Simulation for t=5 seconds**

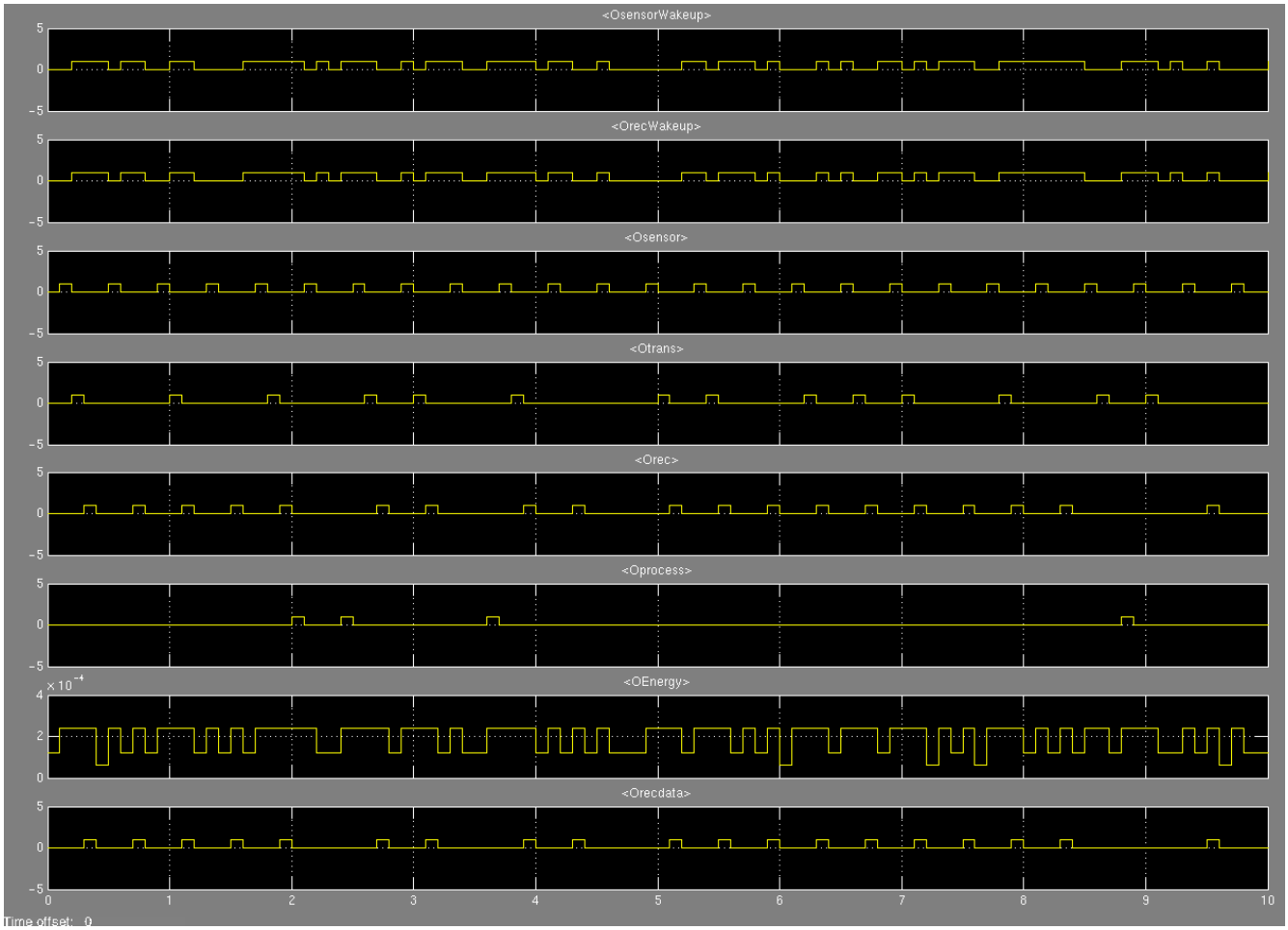


**Figure 5:Simulation for t=10 seconds**

## 7. Conclusion

Energy is a limited resource for wireless sensor networks. Therefore, it is important to evaluate energy consumption which gives an accurate prediction of sensors' lifetime. Sensors are often deployed in remote and/or dangerous areas, it might be very challenging and in some cases impossible to replace batteries.

The model described in this project works based on probability/interruption inputs. A CPU cycle of four basic operations was defined (sensoring, transmitting, receiving and processing) and the model's output represents that definition. This can be confirmed by the results showed in figures 4 and

5. Since the CPU is responsible for sending the data to the transmitter, a data output was added to the model (Otransdata).

# Appendix A: Model Source Code

## Cpumodel.m

```
%CPU POWER CONSUMPTION MODEL
%07/19/2005

function [OEnergy, Orec, Otrans, Osensor, Oprocess]=Cpumodel(Prec, Ptrans, Psensor, Pprocess, DefaultSleepSav,Pm1, PSav1, recTime, transTime, sensorTime, processTime, Complexity, recWakeup, sensorWakeup)

global segment;
%Prec %Probability of receiver awake
%Pm1 %Probability for power mode 1 (1-100)
%Ptrans %Probability of transmitter awake
%Psensor %Probability of sensor awake
%Pprocess %Probability of processing routines awake
%recWakeup %Receiver Interupt, Wakeup CPU
%sensorWakeup %Receiver Interupt, Wakeup CPU
%DefaultSleepSav %Power saving for mode 1, percentage of power used compared to
% full power (1-100)
%PSav1 %Power saving for mode 1, percentage of power used compared to
% full power (1-100)
%Complexity %Choose 0-4 (real numbers are allowed)
Orec=0;%Turn on receiver 0=off,1=on
Otrans=0;%Turn on transmitter 0=off,1=on
Osensor=0;%Turn on sensor 0=off,1=on
Oprocess = 0;
OPower=0; %Output of power average of one cycle (sensor segment, transmit segment, receive segment)
OEnergy = 0;
Tconsup = 0; %Power consumption for current model
CyclePower=0;
CStep = 0;
segmentTime = 0;

%Model Complexity Calibration
```

```
%Full Power Dissipated in Watts
fullPower(1)    = 0;           % Lower Bound of model
fullPower(2)    =24 * 10^-3;    % ATmega128L; Complexity = 1
fullPower(3)    =1.1;          % i386SX-16; Complexity = 2
fullPower(4)    =15.5;         % P54C-200 (Pentium 1-200Mhz); Complexity = 3
fullPower(5)    =115;          % P4-670 (3.8GHz, 2MB L2 - 775); Complexity = 4
fullPower(6)    =250;          % Upper Bound of model

%calculate the full power consumption
n = length(fullPower);

for i=1:n-1
   CStep(i) = (fullPower(i+1) - fullPower(i));
end

complexFrac = mod(Complexity,1);
complexInt = Complexity-complexFrac;

Tconsup = fullPower(complexInt+1) + complexFrac * CStep(complexInt+1);

if(isempty(segment))
   segment = 1;
end

j = mod(segment,4);
if(j == 0)
   j = 4;
end

if(j == 1)
   Prec = 0;
   Ptrans = 0;
   Pprocess = 0;
   segmentTime = sensorTime;
end
```

```
if(j == 2)
   Prec = 0;
   Psensor = 0;
   Pprocess = 0;
   segmentTime = transTime;
end

if(j == 3)
   Ptrans = 0;
   Psensor = 0;
   Pprocess = 0;
   segmentTime = recTime;
end

if(j == 4)
   Prec = 0;
   Ptrans = 0;
   Psensor = 0;
   segmentTime = processTime;

end

p = randint(1,1,[1,100]);
if(Psensor ~= 0)
   if((p <= Psensor || sensorWakeup == 1))
      Osensor = 1
      OPower = Tconsup;
   else
      Osensor = 0;
      if(p <= Pm1)
         OPower = Tconsup * (PSav1 / 100);

      else
         OPower = Tconsup * (DefaultSleepSav / 100);
      end
   end
```

```
end

p = randint(1,1,[1,100]);
if(Ptrans ~= 0)
   if(p <= Ptrans)
      Otrans = 1
      OPower = Tconsup;
   else
      Otrans = 0;
      if(p <= Pm1)
         OPower = Tconsup * (PSav1 / 100);

      else
         OPower = Tconsup * (DefaultSleepSav / 100);
      end
   end
end

p = randint(1,1,[1,100]);
if(Prec ~= 0)
   if((p <= Prec || recWakeup == 1))
      Orec = 1
      OPower = Tconsup;
   else
      Orec = 0;
      if(p <= Pm1)
         OPower = Tconsup * (PSav1 / 100);

      else
         OPower = Tconsup * (DefaultSleepSav / 100);
      end
   end
end

p = randint(1,1,[1,100]);
if(Pprocess ~= 0)
```

```
    if(p <= Pprocess)
        Oprocess = 1
        OPower = Tconsup;
    else
        Oprocess = 0;
        if(p <= Pm1)
            OPower = Tconsup * (PSav1 / 100);

        else
            OPower = Tconsup * (DefaultSleepSav / 100);
        end
    end
end

OEnergy = OPower .* segmentTime .* 10^-3;

segment = segment + 1;

end
```

**CPUInput**

**(included in CPU.mdl)**

```
function [Psensor, Ptrans, Prec, Pprocess, DefaultSleepSav,
                Pm1, PSav1, recTime, transTime, sensorTime, processTime, Complexity] = CPUInput()

Psensor = 100;          %Probability of sensor awake (0-100)
Ptrans = 50;            %Probability of transmitter awake
Prec = 50;              %Probability of receiver awake
Pprocess = 20;          %Probability of processing routines awake
DefaultSleepSav = 50; %Power saving for mode 1, percentage of power used compared to full power (0-100)
Pm1 = 50;               %Probability for power mode 1 (0-100)
PSav1 = 25;             %Power saving for mode 1, percentage of power used compared to full power (0-100)
recTime = 10;           %time in ms for the receiving segment
transTime = 10;         %time in ms for the transmitting segment
```

```
sensorTime = 10;        %time in ms for the sensoring segment
processTime = 10;       %time in ms for the processing segment
Complexity = 1;         %Choose 0-4 (real numbers are allowed) (see below)

%Model Complexity Calibration
%Full Power Dissipated in Watts
%Complexity [0] = 0;                % Lower Bound of model
%Complexity [1] =24 * 10^-3;    % ATmega128L; Complexity = 1
%Complexity [2] =1.1;               % i386SX-16; Complexity = 2
%Complexity [3] =15.5;              % P54C-200 (Pentium 1-200Mhz); Complexity = 3
%Complexity [4] =115;               % P4-670 (3.8GHz, 2MB L2 - 775); Complexity = 4
%Complexity [5] =250;               % Upper Bound of model
```

**CPU Model Wrapper**

**(included in CPU.mdl)**

```
function [OEnergy, Osensor, Otrans, Orec, Oprocess, Otransdata, OrecWakeup, OsensorWakeup] = CPU(Psensor, Ptrans, Prec, Pprocess,DefaultSleepSav, Pm1, PSav1, recTime, transTime, sensorTime,processTime, Complexity, recWakeup, sensorWakeup)

OEnergy = 0;
OEnergySeg = 0;
Osensor = 0;
Otrans = 0;
Orec = 0;
Oprocess = 0;

if(recWakeup > 0.5)
   recWakeup = 1;
else
   recWakeup = 0;
end
OrecWakeup = recWakeup;

if(sensorWakeup > 0.5)
```

```
    sensorWakeup = 1;
else
    sensorWakeup = 0;
end
OsensorWakeup = sensorWakeup;

[energy, rec, trans, sensor, process] = Cpumodel(Prec, Ptrans, Psensor,
                                                 Pprocess, DefaultSleepSav,
                                                 Pm1, PSav1, recTime, transTime,
                                                 sensorTime, processTime, Complexity,
                                                 recWakeup, sensorWakeup);
Osensor = sensor;
Otrans = trans;
Orec = rec;
Oprocess = process;
OEnergy = energy;

if(Otrans == 1)
    Otransdata = 1;
else
    Otransdata = 0;
end
```

**REFERENCES**

[1] O.Landsiedel, K. Wehrle and S. Gotz. Accurate Prediction of Power Consuption in Sensor Networks. University of Tubingen, Germany. 2005.

[2] M. Achir and L. Ouvry. Power Consumption Prediction in Wireless Sensor Networks. Atomic Energy Commission. France. 2003.

[3] V.Shnayder, M. Hempstead, B. Chen, G. Allen and M. Welsh. Simulating the Power Consumption of Large-Scale Sensor Networks Applications. Harvard University.2004.