GREEN LIGHTING SYSTEM

Students
Fulvio Teodori, Stefano Aschettino

Introduction

Power consumption is becoming a real problem in today's world: finding simple and efficient solutions to this problem is probably a challenge that all of us will encounter more and more frequently in our lives. There are, in fact, a lot of situations in which electrical energy is unnecessarily wasted, sometimes for laziness, for not having sufficient care, or simply because people are too busy doing something else.
The aim of this project is to propose an economic and "green" solution to overcome one of these situations, that is to automatically adjust brightness inside a room and keep it always optimal using as more as possible natural light instead of the artificial one.

The basic idea of the system operation is to work on a previously defined range of brightness that represents the "ideal conditions" to aim to: for sure in fact we don't want the brightness to be below a certain level but we don't even want it to exceed a certain threshold, because having an excessive brightness inside the room is first of all useless and could even cause undesired light effects on eventual screens, monitors or other reflective surfaces such as mirrors windows and so on.
The system is equipped with sensors and actuators communicating with him and reacting in order to keep the internal brightness between the specified range, and in a way that minimize as much as possible the power consumption caused from usage of artificial lights.
In order to do this the system always tries always to take advantage of the natural light and turn on the artificial lights only when it is not possible to reach the ideal conditions without doing it (eg during the night). The system controls as well opening and closing of the curtains in the room, in order to modulate external brightness.

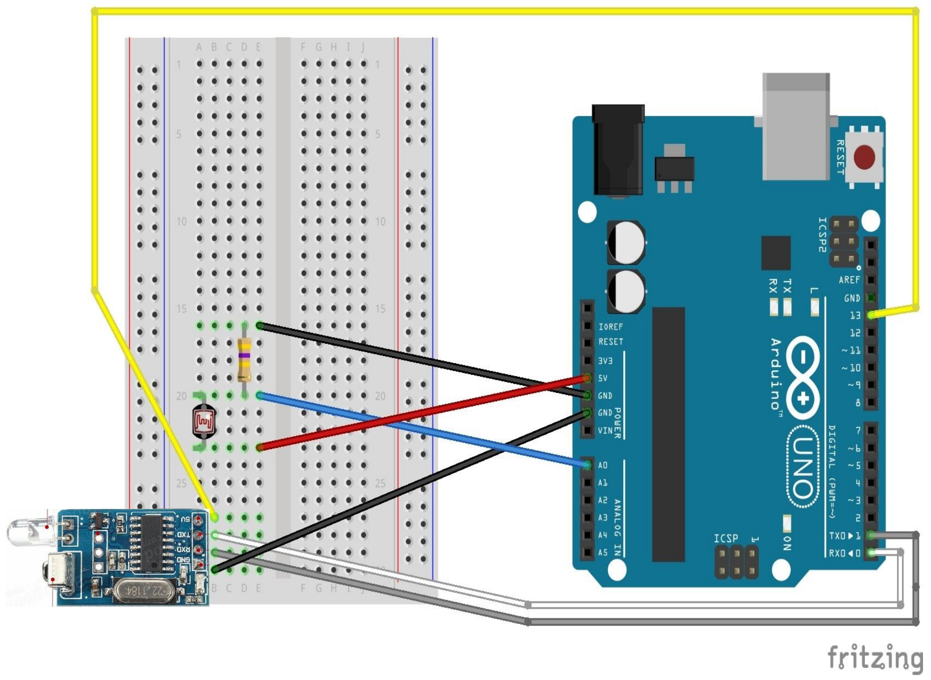
To be thorough it's good to specify that our design choices were made not only with the purpose of minimizing the power consumption of the artificial lights but also the one of the system itself mainly because most of it should be battery operating; this includes the choices concerning the communication networks that will be better illustrated later.

System Composition

The system is composed by different kind of components that are: internal light sensors, external light sensor, central system, curtain and lamp controllers.
In following sections we'll illustrate them one by one analyzing hardware specification and basic behaviour of all of them.

Internal sensor

Internal sensor of the system is a component of a sensor network properly distributed in the room providing internal light conditions to the central system (in our case there are 3 internal sensors).
Each internal sensor periodically reads the value of the brightness, adapts it in a range between 0 and 100 and sends it via IR to the following node of the ring; last node instead send all received data and to the central system via a cabled software serial connection with it.
A more detailed description of the IR network properties will be provided in the related section.

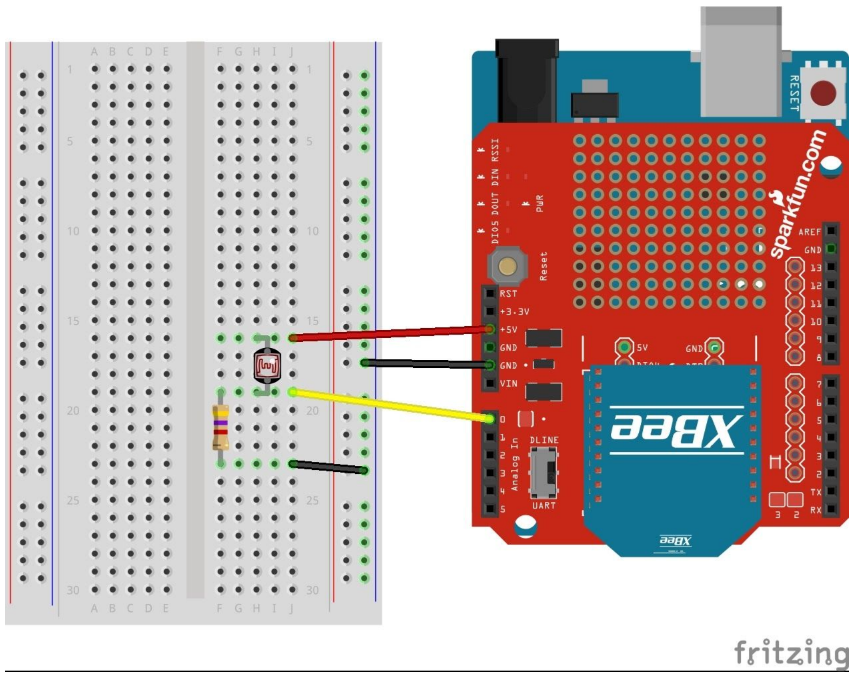


Each internal sensor is composed of:

- an Arduino Uno;
- a GL55 photoresistor;
- a 470kΩ resistor;
- a NEC infrared module YS-IRTM.

External sensor

The external sensor is a wireless component of the system whose role is to provide natural light conditions to the Central System
It periodically reads the value of the brightness, adapts it in a range between 0 and 100 and sends it to the central system via a Zigbee module.



The external sensor is composed of:

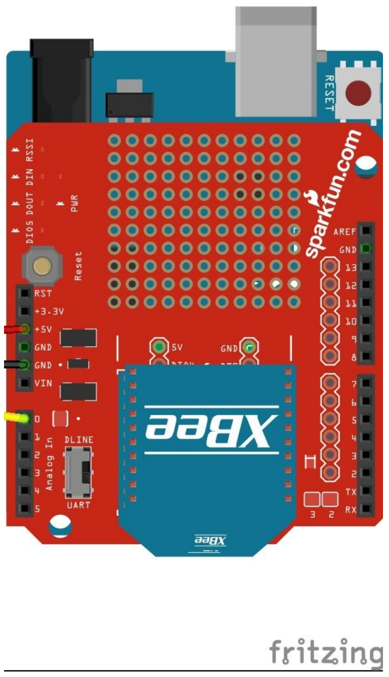
- an Arduino Uno;
- a GL55 photoresistor;
- a 470kΩ resistor;
- an XBee (Series2) Wireless Communication Module;
- an Xbee Shield V03 Module.

Central system

The central system is the component of the system that collects data received from both external and internal sensors and transmits proper commands to actuator controllers.
It communicates through a Zigbee module with external sensor and actuators and via a cabled software serial connection with the last node of the internal sensor network. Once received light values from sensors, it elaborates them and, if needed, sends proper values to both the actuators.

The main logic of the central system is based on a range of proper light conditions defined from RANGEMIN and RANGEMAX parameters then the system operate as follows:

- If the internal brightness is within this range and the artificial lights are used their values is decreased by a minimum;
- If the internal brightness is too low system tries to increase it by opening the curtains in the room; if instead curtains are already fully open or there is not enough brightness outside, then increase artificial lights is the only way to re-establish proper conditions and it’s done.
- If the internal brightness is too high then system decrease it by dimming or fully turning off artificial lights; if lights are already off, then gradually close the curtains.



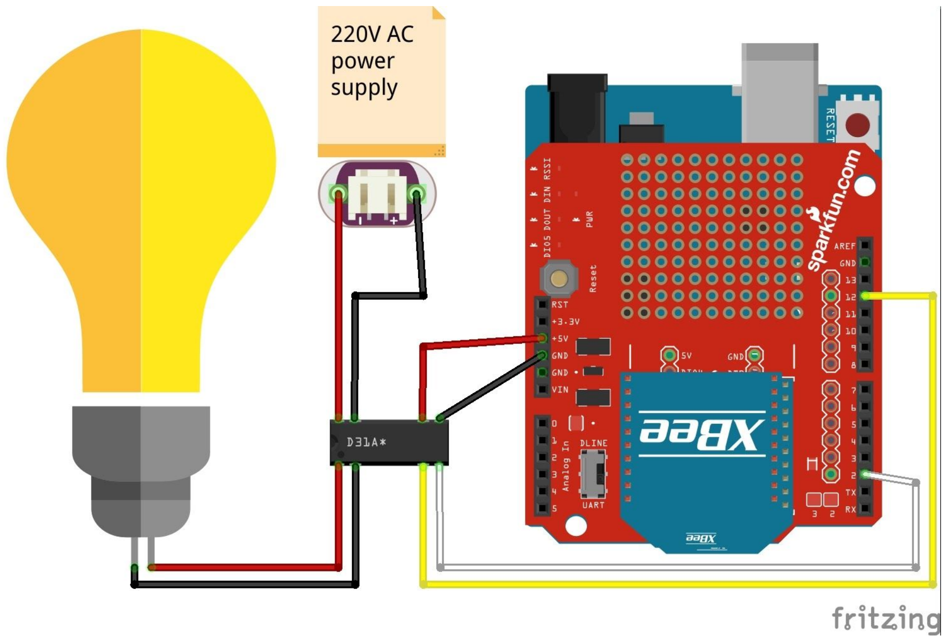
The central system is composed of:

- an Arduino Uno;
- an XBee (Series2) Wireless Communication Module;
- an Xbee Shield V03 Module.

Lamp controller

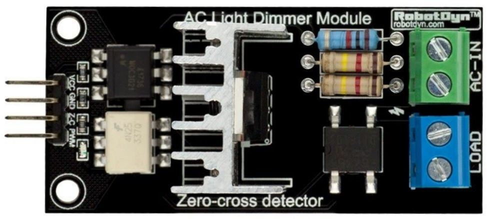
Lamp controller is the component of the system in charge of modulate intensity of artificial light, or eventually switch them totally off, according with values received from central system.

Its main role is to control current flowing from the wall using an AC dimmer module that will properly modify the sine-wave received from the load (in our case a simple light bulb that should simulates the whole room lighting system).



Lamp controller is composed of:

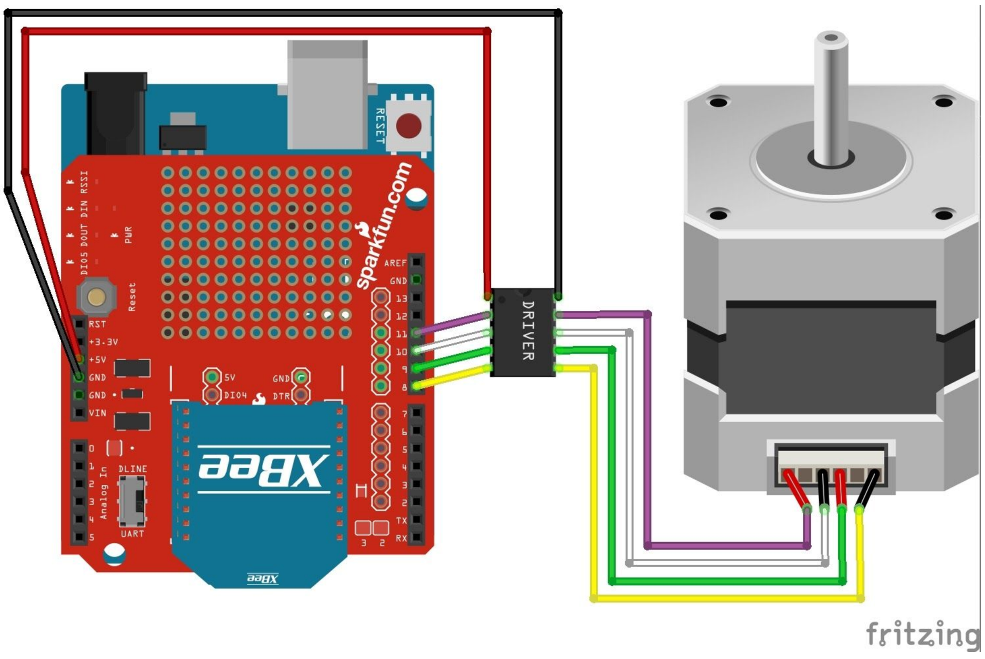
- an Arduino Uno;
- a Robotdyn AC dimmer module;
- a light bulb;
- an XBee (Series2) Wireless Communication Module;
- an Xbee Shield V03 Module.



Robotdyn AC Light Dimmer Module

Curtain controller

Curtain controller sets the curtain to the level defined from the central system by opening or closing it.
It receives data from the central system via a Zigbee module and reacts considering the current position of the curtains and then calculating and giving to the stepper motor the correct command (in terms of both steps number and direction) that will finally move the curtains.



Curtain controller is composed of:

- an Arduino Uno;
- a 28BYJ-48 stepper motor;
- a ULN2003 driver module;
- an XBee (Series2) Wireless Communication Module;
- an Xbee Shield V03 Module.

Design and Implementation

For what concern design and implementation of the system the largest part it's not too much complex and then can be easily deduced also by reading the code.
On this section will be analyzed instead two important aspects on which it's worth to focus a bit more that are the ones related the communication of the project and more specifically the design of the Infrared and Zigbee networks.

Infrared Network

The IR network of our system combines in it features from different existing technologies, we can say in fact, with a good approximation, that it consists in a logical ring network using a TDMA approach.
These features are mainly caused from inbuilt characteristics of the infrared technology: in fact for its nature IR communication is basically a broadcast one (unless you provide addressing at an higher level) and all the IR-communicating nodes share the same media, furthermore IR always needs a clear line of sight between communicating nodes in order to properly work.
In particular, TDMA approach is useful to ensure that only transmitting and receiving nodes are running during transmission. In fact, according to the above (and always assuming no addressing provided) each other node running at the same time and 'seeing' the communicating ones could think to be the intended recipient of the message.
The ring topology is instead necessary in order to provide scaling properties to the network, in fact thinking to an increasing number of nodes in the same room is difficult to believe that all of them has a clear line of sight with each other, it's easier instead to set them up in order to see 2 by 2. According to this property and the fact that all the nodes need, sooner or later, to carry its own messages to the same destination, the most appropriate solution seemed to be a ring topology with a chain of messages. In other words basic behaviour of each node is to receive messages from previous node and transmits all of them plus its own one to the subsequent node.
Obviously the main difficult about implementing our IR network has been to synchronize each node's uplink. In fact, considering the nature of the project and the fact that all the IR nodes (except the last one) should be battery operating, an issue was to provide the less overhead possible to the node's execution in order to keep the total uplink time low and making batteries last more. Due to this the chosen solution was to synchronize timing between nodes by estimating the execution time of each one. Here below we'll explain how this estimation has be done.
As stated above each node on his turn should receive messages, read its own value and send everything, but considering that IR modules we used need a quite relevant

startup time instead time for reading the LDR is negligible we can modify the previous steps and write as follows:

UplinkTime = IRstartupTime + RecTime + SendTime

According to the above and looking at the whole network we should also consider that the SendTime of each node match with the RecTime of the subsequent one and obviously that first node doesn't receive anything; then taking into account the goal of minimizing the total uplink time we have that the optimum total execution time should be:

TotalExecTime = IRstartupTime(0) + SendTime(0) + SendTime(1) + …+ SendTime(n)

Analyzing previous statement can be noted that only startup time of first node is considered this implies that startup of other nodes has to be made during other operations because otherwise each node should wait for startup of next one resulting in an energy waste. Starting from this consideration we know that at the beginning of each loop each node has to wake up after the RecTime of all the previous ones, in fact for each i we have:

wakeUp(i) = wakeUp(i-1)+*IRstartupTime(i-1)+RecTime(i-1)-IRstartupTime(i)*

that is:

wakeUp(i) = wakeUp(i-1)+*RecTime(i-1)*

and then:

wakeUp(i)=wakeUp(0)+*RecTime(1)+...+RecTime(i-1)*

Having to calculate the statement above we considered that recTime consists in receiving one message from each previous node, then being messages of the same size we started from an estimated transmission time (msgTime) and we have that:

*RecTime(i)=msgTime*i*

making same reasoning for SendTime we have that the uplink time can be rewritten as follows:

*UplinkTime(i) = IRstartupTime + msgTime*i +msgTime*(i+1)*

Due to the above we see that uplink time is different for each node then we can guess that in order to keep the same distance between nodes' wakeUp at each loop we can't make everyone sleeping a fixed amount of time unless we add some kind of "padding" simulating that all the uplink have the same duration.

This padding is actually a dynamic delay added to the fixed one and can be calculated in a recursive way, as already made for distance, starting from the fact that uplink of each node is greater than the previous one by two msgTime at the end we have that:

*padding(i) = 2 * msgTime * (n-1-i)*

All the assumptions made above have a clearly theoretical nature and in order to translate them in lines of code are often needed some adjustment mainly due to the fact that hardware behaviour could never be as regular as theoretically planned and

can also change according to which component you are using.
In our case these elements makes necessary to introduce two exceptions to the previous hypothesis: a delay between reception and transmission of each node in order to space out the uplink of the nodes (lazyTime); and a different duration for the first message transmitted probably due to some initialization of IR module (firstSend).
Once ensured that timing between nodes is correct and is maintained during the execution the last issue was to provide the same time0 to all the nodes that being independent one to another is not so trivial as it seems.
In order to do this we implemented a network startup phase during which in contrast to the rest of the operation all the IR modules are awake waiting for a start message that will represents their time0. Once received this start message (specific for each node) each node generate the start message of the subsequent one and transmits it, then go to sleep and wake up for its first uplink.
This is possible because estimating the execution time of this phase and knowing (according to previous explanation) the distance needed from previous node each node can calculate how much time to sleep being sure to wake up at the right time for receiving messages from its predecessor.

Zigbee Network

Before using the Zigbee modules for data exchanging among sensors, central system and actuators, each module needed to be configured in a specific way.
First, we had to choose the network topology: a star topology with the central system in the center was the obvious choice.
Then we configured each module using a usb adapter (dongle) and a specific software named XCTU.
In order to set up a Zigbee network, each module must be configured with a Radio Channel (CH) and a Personal Area Network identifier (ID); only modules with the same CH and the same ID can communicate with each other.
The central system was configured as the Coordinator of the network, setting the Coordinator Enable (CE) parameter to 1, while the other Zigbee modules for sensors and actuators were configured as End Devices, setting the CE parameter to 0.
The coordinator is located in the center of the star topology and connects to every end devices.
In this type of configuration, end devices do not communicate with each other (there’s no need to do that in our system): every message must pass through the coordinator.
Given the “green” approach and the non-critical nature of the system, we chose to manage the communication among the Zigbee modules with the so called Transparent Mode, setting the API Enable (AP) parameter to 0.

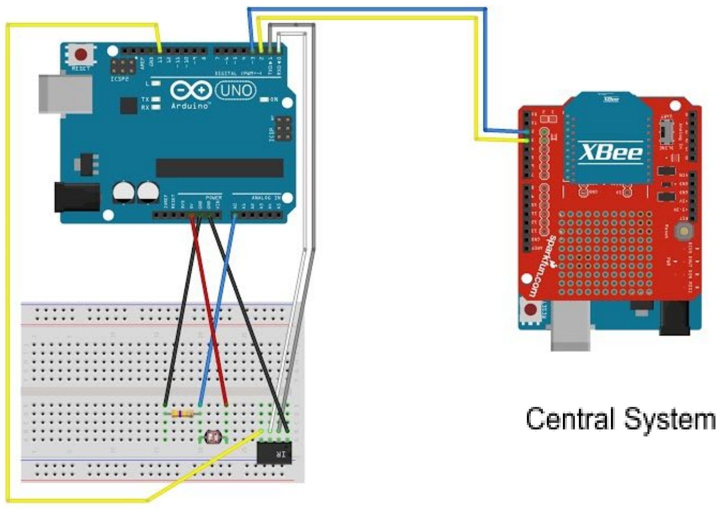
The transparent mode is designed to replace a serial connection among Zigbee modules, a connection that is simply raw data flowing into the radio channel.
In this way, we removed the overhead of the communication typical of the Zigbee's API mode, in which data are structured in well-defined frames with a start delimiter, a frame length, a frame type, a checksum and so on, thus minimizing the quantity of data exchanged and hence the power consumption.
Data are sent in broadcast, thus avoiding the need to manage the bulky 64 bits addressing of the Zigbee protocol.
At the recipient's side, the verification if the received data are destined for that specific node, is carried out at user application layer; besides, when receiving broadcast messages, modules do not send acknowledgements by default, thus avoiding additional data to be transmitted.
Given that our system's core algorithm works sending and receiving integer values from 0 to 100, we chose to handle the communication among the Zigbee nodes with a simple custom user application layer protocol.
The protocol works as follows:

- Messages exchanged among nodes have a fixed size of 3 bytes;
- The first byte is a control byte used, along with the second byte (described later) to avoid potential unexpected values to reach the actuators, unexpected values that could be caused by the configuration messages autonomously exchanged among the Zigbee nodes during the initial network setup;
- The second byte is used to identify which end device is to be considered the recipient of the message or which end device is the sender of the message.
  Let's say that in our system there is just one Zigbee end device (the external sensor) which sends messages to the central system, so the latter could always know which is the sender of the message even without this identifier; we simply chose to be consistent with the size of the messages exchanged among the nodes and we also chose to define a protocol that can be scalable with the potential introduction of other sensors.
- The third byte is the actual payload, that is the value from 0 to 100 used by the system's core algorithm to control the brightness inside the room.

Conclusions

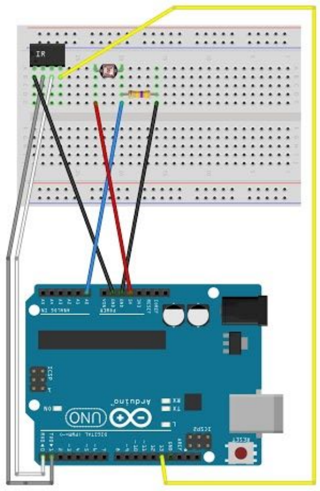
The main purpose of the project was to develop a system able to regulate brightness inside a room and to do that according to a “green” philosophy; we can say at the end of this experience that this purpose has been someway reached.
Anyway while working on the system we noted that will be probably some possible further development that could be of a particular interest, like for example an integration with a complementary system taking care of other conditions inside the room and adopting the same approach (eg heating system) or an integration with an higher level application that allows the definition of some “scenarios” that will automatically modify the range of ideal light conditions (eg lower brightness for watching a film or making a presentation, higher brightness for reading newspapers and so on).
In conclusion we can say that development of the system allowed us to acquire more skills especially related to communication and synchronization at such a low level that, being a new experience for both of us, has represented at the same time the biggest challenge and the more fascinating aspect of the project.

Appendix A - Overall View of the System

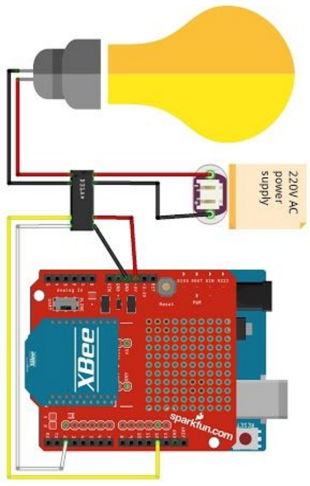


Internal Sensor 3

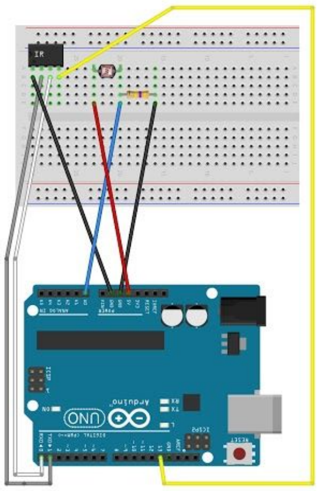
Central System



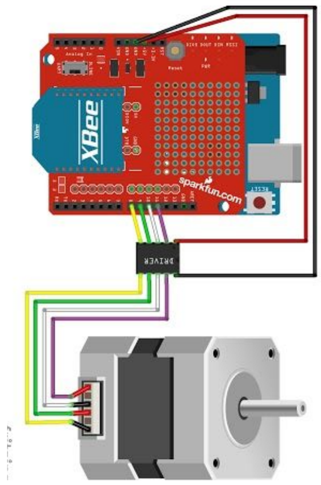
Internal Sensor 1



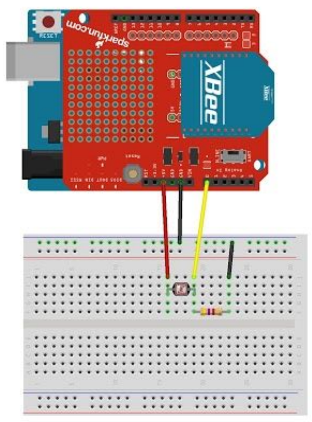
Lamp Controller



Internal Sensor 2



Curtain Controller



External Sensor