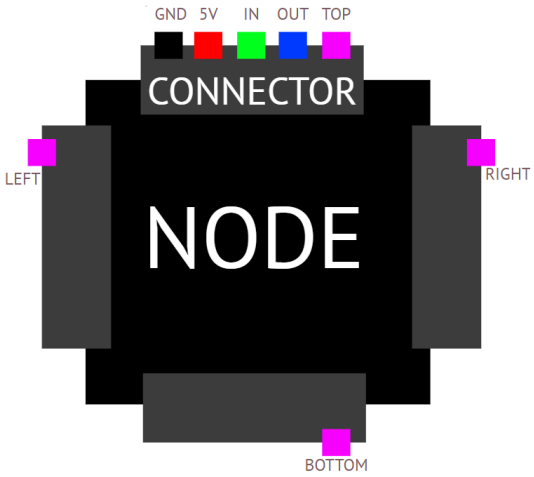
Physical setup:

A node has a total of four connectors one on each side.

A connector has five pins in it: input and output pins for communication, ground and 5v for power and a direction pin. For the configuration checker only the direction pin is important, and it is the only pin which is unique.

Two nodes can be connected side by side. This connects the same pins together, GND to GND and so on. With direction pins the connections are always either TOP – BOTTOM or RIGHT – LEFT. The configuration method does not need this property.



Packet sending:

A packet is sent with the PJON SoftwareBitBang protocol [https://www.pjon.org/SoftwareBitBang.php].

packets sent by the configuration method are byte (uint8_t) arrays with the first byte being a packet id. The configuration method sends five different packets:

1. Master broadcasts* a start signal (ID ‘L’)
2. Master broadcast a stop signal (ID ‘S’)
3. Master broadcasts an id which is only accepted by one slave (ID ‘I’)
4. Master sends a direction packet to a specific slave (ID ‘N’)
5. Slave replays to id broadcast with its information (ID ‘O’)

* Sends packet to everyone in the configuration.

The configuration method sends only blocking packets, meaning that a packet is repeatedly being sent until it is received or a maximum amount of retries is reached. This could block the program for up to 4 seconds in case of a failure. Blocking packets are necessary because a direction is declared ‘empty’ if no answer is received from that direction: If a packet is lost, then no answer will be received, and that direction would be falsely declared empty.

To receive a packet, the received must be in the reception-state, this is accomplished with a receive function which takes a timeout as an argument.

Calculating slave positions:

For determining positions of the slaves, the method uses a stack of pairs which is implemented with a struct array called *stack*. The struct *stackPair* consist of a node id and the next direction it should check, both are bytes. We keep track of the node highest in the stack in its own variable called *stackTop* which points to the index of the highest node in the stack. Initially the *stackTop* is equal to zero pointing at the master with the masters next direction being 0 or *TOP*. Every new node entering the stack starts with the direction 0.

A slaves position is calculated when it is found by going through the stack and checking the directions we have taken. In example if the current stack is {0: LEFT, 3: TOP, 4: TOP, 6: BOTTOM} and a new slave is found it must be in position LEFT TOP TOP BOTTOM or 3002. The stack is then updated. This direction is saved as a string.

Configuration method:

The configuration method is used to find the location of each node. In addition, it is used to assign each node a unique id.

The method is based on setting the direction pins high one by one for each node and waiting for responses from the receiving side. This means that the configuration method could be split into two parts, one for the master and other for the slaves. The master is mainly responsible for doing the configuration and all variables mentioned are on the master side if not mentioned otherwise.

The stack mentioned in previous section is the building block of this method. The stack is updated on every call and is responsible of keeping track of the current position. The stack is a local variable which is passed to the method every call instead of being a global variable. This is to save some memory.

To assign a unique id to each slave we use a running integer starting from 1 which can go up to 253. We call this *nextId*. Id 0 is reserved for broadcasting and 254 for the master. This means that the configuration can have up to 253 nodes excluding the master.

In addition, we use some flags to avoid certain situations:

Both the master and the slaves have an *isConfigurationDone* flag, which is used to stop executing the receive function and other functions before configuration is done.

The slaves have a *waitingForSignal* flag which is only true when in the configuration state and the slave has not received a signal. When a slave receives pin signal in the configuration state for the first time 'waitingForSignal' flag will be set back to false to avoid loops in the configuration.

The *isCurrentReceiver* flag is for assigning unique ids to the slave as the master is initially unable to send packets to a specific node. With this flag the master can broadcast a packet and only the slave with this flag up will be able to know that the packet is meant for it.

The method starts when the master reads the byte 'R' from the serial port. The master then sends the start signal to each node in the configuration. The nodes react by setting the 'waitingForSignal' flag to true and all their direction pins to input mode.

Now we proceed to the actual configuration method: First we check the next direction of the *stackTop*, we call this *dir*. Next, we check whether the master is on top of the stack or not by checking the *stackTop* integer, here 0 would mean that master is at top. The next steps are slightly different for the master and the slaves but in both cases, we check whether *dir* is valid, meaning that is either 0, 1, 2 or 3:

- Master:
    - *dir* is valid: Set the pin in *dir* to high and all others to low
    - *otherwise: Master has checked all its directions -> The configuration is done, STOP by setting the isConfigurationDone* flag to true and broadcast a packet informing the slaves that the configuration is done.
- *Slave:*
    - *dir* is valid: Send a packet with *dir* to this slave. The slave will then set pin in *dir* to high and all others to low

- otherwise: Slave has checked all its direction -> We can remove this slave from the stack and move to the one below it by decrementing *stackTop* by one and RETURN -> The configuration method calls itself so the execution will be continued from +

Now the master or a slave has set their pin in *dir* to high. If a slave, with *waitingForSignal* flag being true, notices that a pin is set to high, it reacts by setting the *waitingForSignal* flag to false, the *isCurrentReceiver* flag to true and sends a packet (In current implementation the packet contains information of the slaves connected components) to the master. It is important to notice that no packet is send if no slave received the signal or is not waiting for a signal. This is because next the master checks whether it has received a packet (the master only waits 0.05 seconds for each packet):

If a packet is received, then a new slave has been found: We calculate the position of this new slave and add it to the stack (In current implementation the master prints to serial information* of this node). Then the master broadcasts an id and increments both *nextId* and *stackTop* by one. Every node will receive this id packet but only the slave with the *isCurrentReceiver* flag set to true will set the received id as its own id. After receiving the id, the node sets the *isCurrentReceiver* flag back to false. We call the configuration method with the updated stack.

+ Finally, we increase *dir* by one (and update the stack) and call the configuration method.

*Id, position, and count of each component. This way we can save some memory as we do not have to save the information on the board.

Connecting and disconnecting nodes:

Master should only disconnect when it is unplugged from the USB-port. This should be handled on the pc side

Whenever a node connects it will send a packet to the master which is only received in the configuration is done. This way the master will know if a new node has connected.

For disconnecting nodes, we use the direction pins again: In the configuration state if a node finds another node next to it, the master will inform about this to the node and the node flags that direction. After the configuration state all nodes set the flagged direction to input and all other to output a high voltage. Now if a flagged direction reads low a node in that direction has disconnected.