Simple Serial Bus

A New Serial Interface Concept

By James H. Reinholm

1. Introduction

The development of digitally based products has experienced phenomenal growth worldwide in recent years, and it appears that this growth will continue to increase at a rapid rate. Various industries such as telecommunications, data processing, and home entertainment systems rely heavily on the use of this new digital technology. Current integrated circuit technology, namely VLSI (very-large-scale integration), has made it possible to develop special purpose digital devices and systems that are capable or performing a wide variety of complex digital processing operations. Some of these processing tasks are many orders of magnitude faster, smaller, and cheaper than they were ten years before. Also, many functions that were usually performed by analog means are now realized by less expensive and more reliable digital hardware.

Most of these devices and systems require a form of communication to transmit information to and receive information from another system or device. As an engineering team goes through a typical development process for a system or device, they don't think much about the communication aspects of the system, but instead they usually focus on the basic functionality from a user point of view so that the unit would operate as intended when it is plugged in or turned on. The engineer in this case would come up with a custom designed module from a selected group of standardized components and possibly from other components that are designed and built from scratch. The 'scratch' components would be anything that is non-standardized and could be just made from raw materials. This could also be a new idea or concept that has not been tried before. Setting up a new kind of configuration assembled from standardized components could also be included in this category.

The standardized components would include any part with a specification attached to it, such as integrated circuits, transistors, resistors, etc. This standardized components category would also include non-physical objects such as programming languages (C++, Java, Perl, etc.). There are also standardized routines within these programming languages along with several hardware and software combinations that can also be considered as standardized components. The communications protocols and networks can also be included in this category, and these would include well known standards such as RS-232, RS-485, USB, SPI, I2C, and many others.

As the engineer moves along in his design process, and he is trying to come up with a system that would satisfy the customer, he wouldn't waste much time in developing a new communication protocol, but would instead use one of the established methods and make adjustments to his system design so that it fits in. The most likely choice would be USB, since this is such a universal protocol, and many existing devices communicate using this protocol (printers, scanners, cameras, etc.). Because of the popularity and other advantages of the USB interface, some other interface protocols, such as RS-232 and RS-485 are becoming obsolete. Even though USB is simple to connect, the engineer would soon discover many features about USB data packet transmission that are very difficult to understand.

To get some idea of the complexity of the USB protocol, it would be helpful to explain some of its communication methods. When a USB peripheral device is connected to the host, a process called the enumeration process is started. This is where the peripheral sends information to the host about its identity, device drivers needed, device speed, address, etc. This happens every time a device is connected or disconnected.

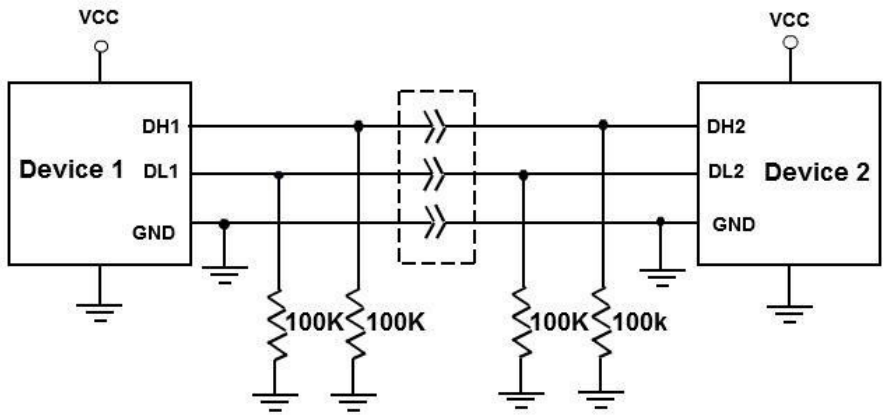
All data transfers between USB devices occur through virtual ‘pipes’ that connect the peripheral's addressable ‘endpoints’ with the host. An endpoint is a uniquely addressable portion of the peripheral that is the source or receiver of data. When establishing communications with the peripheral, each endpoint returns what is called a ‘descriptor’. A descriptor is a data structure that describes the endpoint's configuration and expectations, and include transfer type, max size of data packets, perhaps the interval for data transfers, and in some cases, the bandwidth needed.

Sometimes an engineer would see that many of these advanced features would never be used on his project, in which case he would decide to use RS-232 or RS-485 instead. Even the RS-232 and RS-485 interfaces have extra features that would probably not be needed and lack certain qualities that would be needed, such as reliability in a noisy environment and self-adjusting capability. Some of these older protocols also lack reliable methods for detecting connection present and certain errors that occur during data transfers. I will not discuss the particular features of RS-232 and RS-485 at this time, but I will leave that for discussion later on.

There are also instances where hobbyists are designing and building a simple digital/analog circuit and need an effective method of communicating with another device without getting involved with the complexities of USB. An example would be a weather indicator, which transmits temperature, barometric pressure, wind speed, and wind direction to a separate display device. Another example would be for an exercise machine or a bicycle, which can transmit parameters like speed, cadence, time used, and distance traveled to a recording device.

The aim here is to come up with a simple serial interface or SSB (Simple Serial Bus) that would replace some of the more obsolete methods such as RS-232 and RS-485 and to compete directly with the established USB interface in terms of increased simplicity and reliability, particularly in error detection. In designing such an interface, it is important to consider what the engineer or hobbyist actually needs in order to implement serial data communication in his project. He would usually be operating on a limited budget, so he would probably need something that he can understand easily and come to a quick determination as to whether it would perform the needed data transfer in a reliable manner.

In order to begin the design process of such an interface, it would help to have a simple block diagram as to how the system would appear to the engineer as he looks for a suitable communication protocol for his project. I would start with something like:



This diagram attempts to simplify things by using two tri-state bi-directional data lines along with a ground signal through the connector. The simple arrangement of components as shown would be a good starting point that could be use to maximize reliability for a basic serial communication system.

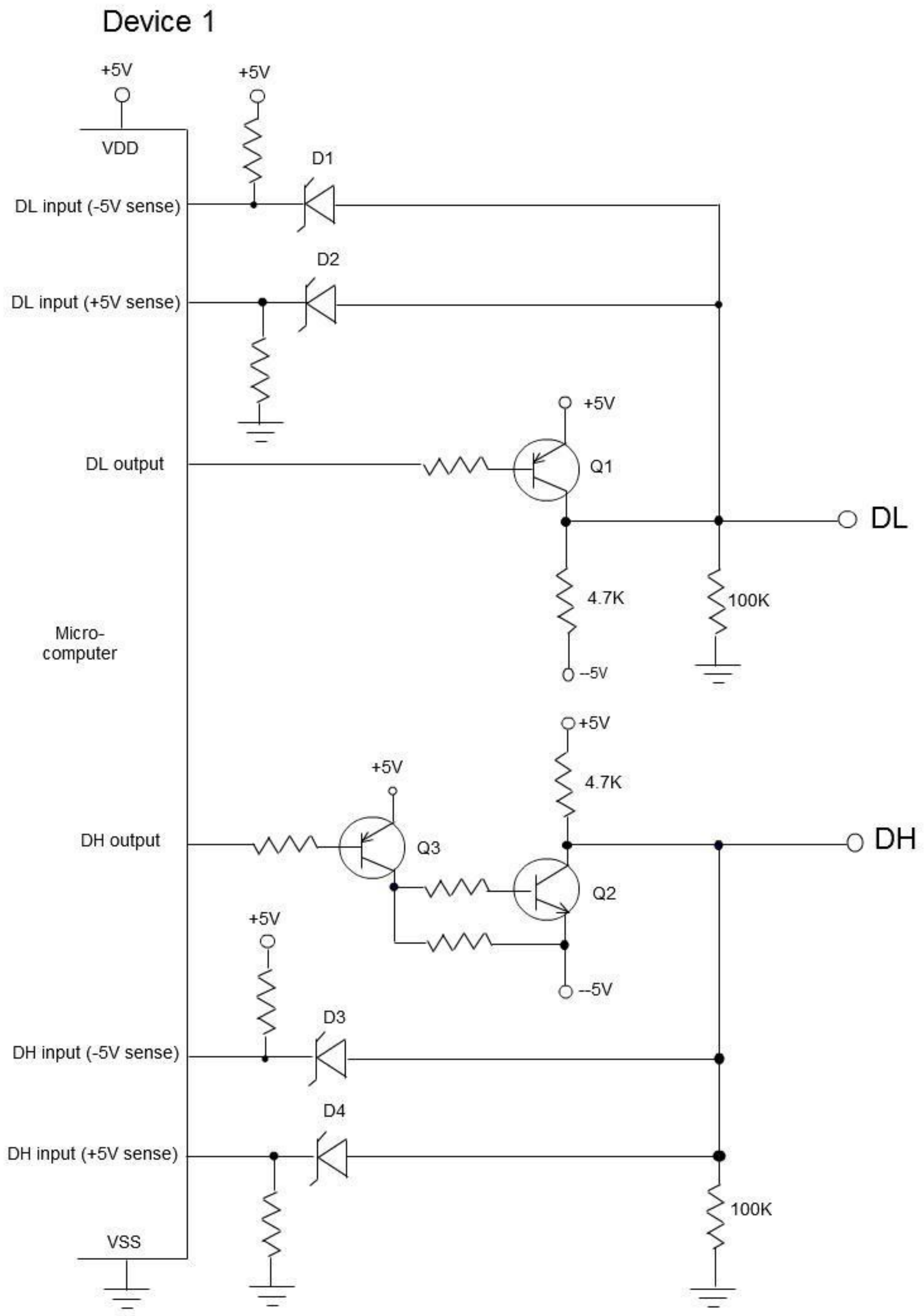
The main advantage that the Simple Serial Bus would have over other implementations, such as USB, would be its inherent simplicity, and the concept would allow easy inclusion into low-level microcontroller systems in a similar fashion that standards such as RS-232 and RS-485 have done in the past.

2. Data Line States

The DH (data high) and DL (data low) would be set up as two interacting data lines that are mirror images of each other. There would be an idle state or condition where no data transmission is taking place where the DH line is pulled to +5V (logic ’0’) on both sides by internal pull-up resistors and the DL line is pulled down to –5V (logic ‘1’) on both sides by internal pull-down resistors. As long as the connector is plugged in and both devices are active with power applied, each one of these lines can be considered as inputs and outputs on both sides simultaneously in the idle state. The internal pull-up resistors provide the mechanism where both sides can be outputs at the same time without damaging the devices by excessive current drawn. Normally, in the idle state there wouldn’t be any current drawn through these resistors anyway because each connected device would pull the line to the same level.

This technique adds a great deal of simplification as compared to other serial communication circuits where they avoid having both sides used as outputs at the same time by a multiplexing arrangement or other logic circuitry. When one end (transmitter or receiver) is used an output, the other end would have to be set up as an input. Both sides set up as inputs would also be allowable, but both sides could not be used as outputs simultaneously because of the push-pull output stage and there would be excessive current drawn if one is high and the other low.

The DH and DL lines also have three other possible states other than the ‘idle state’ just described. These are the ‘active state’ and two ‘zero’ state modes. The ‘active state’ of DH uses a transistor switch which applies -5 volts directly to the DH line, and the active state of DL uses another transistor switch which applies +5 volts directly to the DL line. The following circuit diagram will show what has been described so far for the active and idle states. This circuit is just a basic starting point and more features will be added later as developments are made. The circuit shows just Device 1 instead of both devices for simplicity.

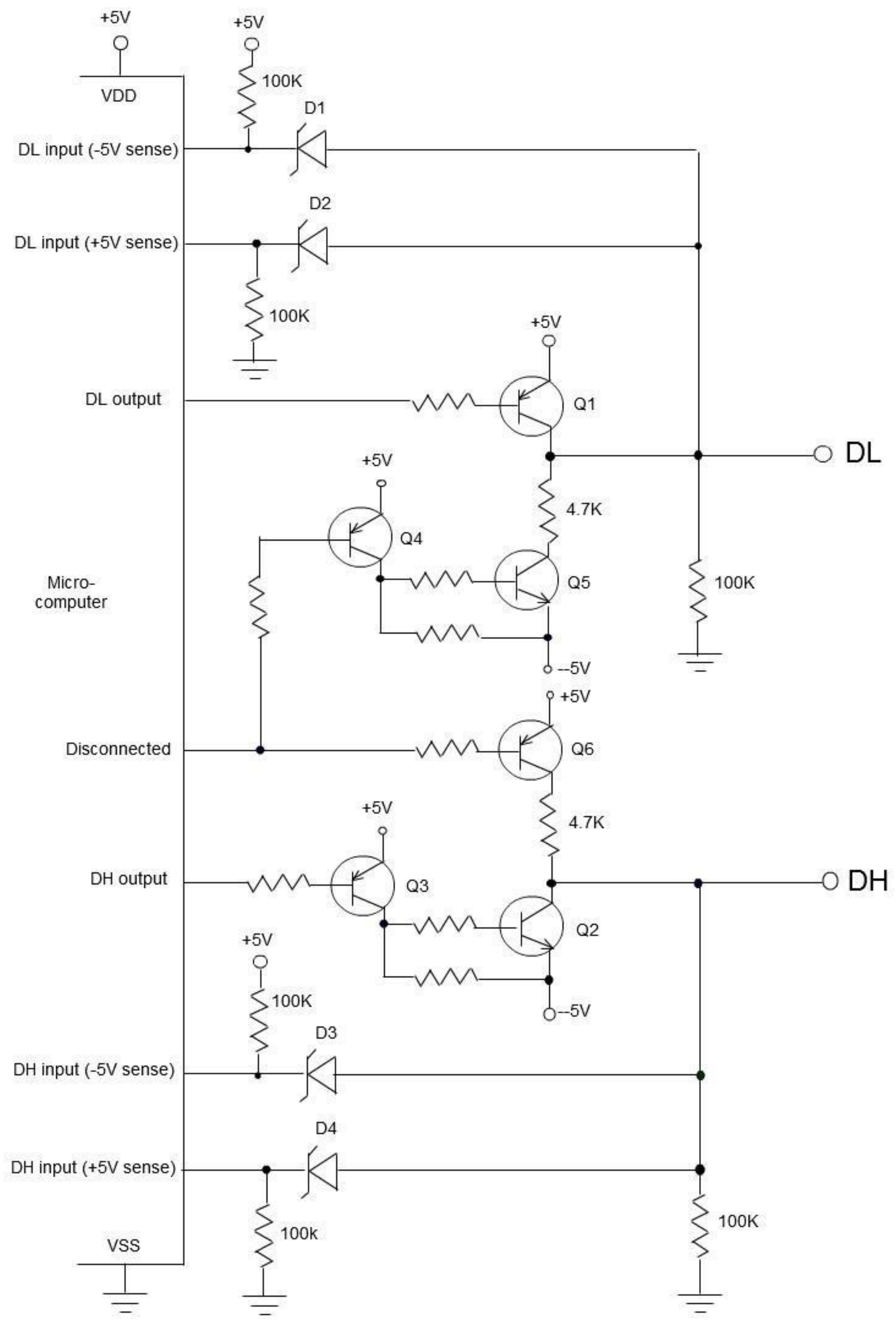


Since the data lines operate from +5 to –5 volts, and the microcomputer operates only from +5 to 0 volts, there are level shifting components needed. Q3 is used for the DH output and D1, D2, D3, and D4 are used for the inputs. There are two inputs needed for both DL and DH to detect the full range from +5 to –5 volts.

This basic circuit would need additional components in order to use the two ‘zero’ state modes on the data lines. The DH and/or DL line is set to ground potential (0 volts) in each of the two ‘zero’ state modes, where one is a ‘strong’ ground and the other is a ‘weak’ ground.

The strong ‘zero’ state mode is mainly used for issuing break requests and is called break-request-high (BRH). The weak ‘zero’ state mode is called break request-low (BRL) and is used mostly as a low-power standby feature where a device disables its driver circuitry in networks when there is data transfer taking place and it is not on either transmitting or receiving end. It is also used when a device has sensed a disconnection, or if the devices (or just one device) are turned off at the opposite end, or if there is a fault in the circuit. In this state, the pull-up or pull-down resistor is effectively disconnected and the active voltage supply switching transistors are off, and creates a high-impedance input circuit on the data line. This state would be typical of a low-power mode, and this ‘zero state’ mode could also be activated if it is desired to conserve power in certain time periods when there will be no data transmitted or received. The following diagram will show how the weak ‘zero’ state mode would be added to the previous circuit. The BRL signal is controlled by the ‘Disconnected” pin which disconnects the +5 and –5 volt supplies from the driver circuitry by turning Q5 and Q6 off. In normal operation, Q5 and Q6 are kept in the ON state. Q4 is just a level shifting transistor.

Device 1

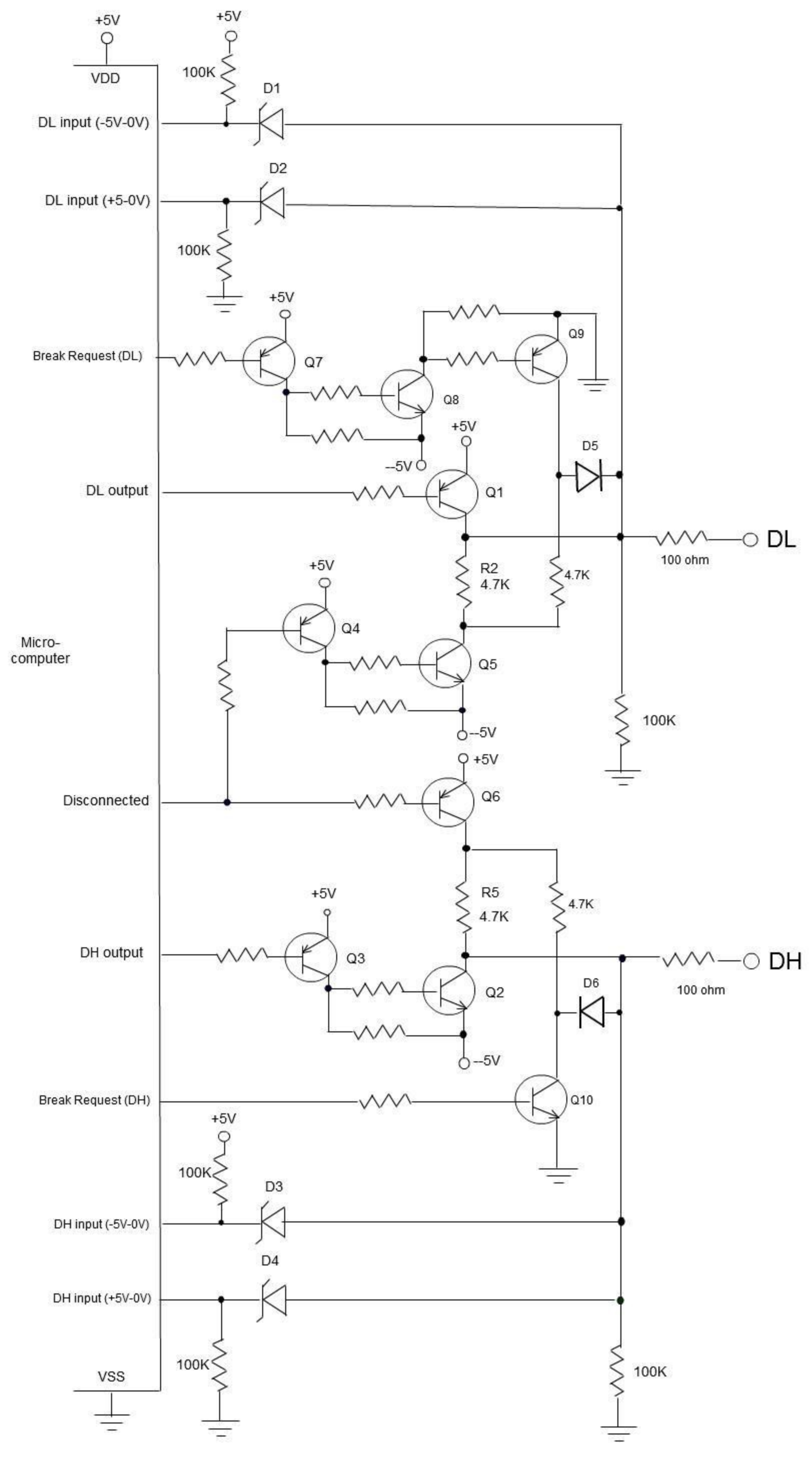


Note: All resistors not labeled are 15K

The two data lines are independently controlled in the data transfer process, so it’s possible for both lines to be idle, or one line idle and the other active, or both lines active at the same time. When a line is made active on a transmitting device, it can only be an output and not used as an input unless an “emergency break request” is issued by another device or a fault condition is detected. The device (or devices) on the opposite receiving end will immediately become inputs which will detect the presence of an active data line. This “emergency break request” is a new feature recently added to for high priority interrupts, and is rarely used compared to the standard break request.

Since the idle state is determined by the voltage through a pull-up or pull-down resistor, and an active state sets a data line to +5 or –5 volts directly, the active state always has priority over and overrides the idle voltage state. The break request-high (BRH) is an interrupt feature which can also override the idle voltage and bring the voltage of the data line to ground potential (0 volts) directly through a switching transistor. The following diagram shows how the circuit would look with this feature added. Q9 and Q10 are the switching transistors that pull DH and DL to ground from their idle states. D5 and D6 prevent interference between the two active transistors on a data line. This circuit does not include the rarely used “emergency break request” feature, so the ordering of the four possible states so far would be (1) active, (2) break request-high, (3) idle, and (4) break request-low.

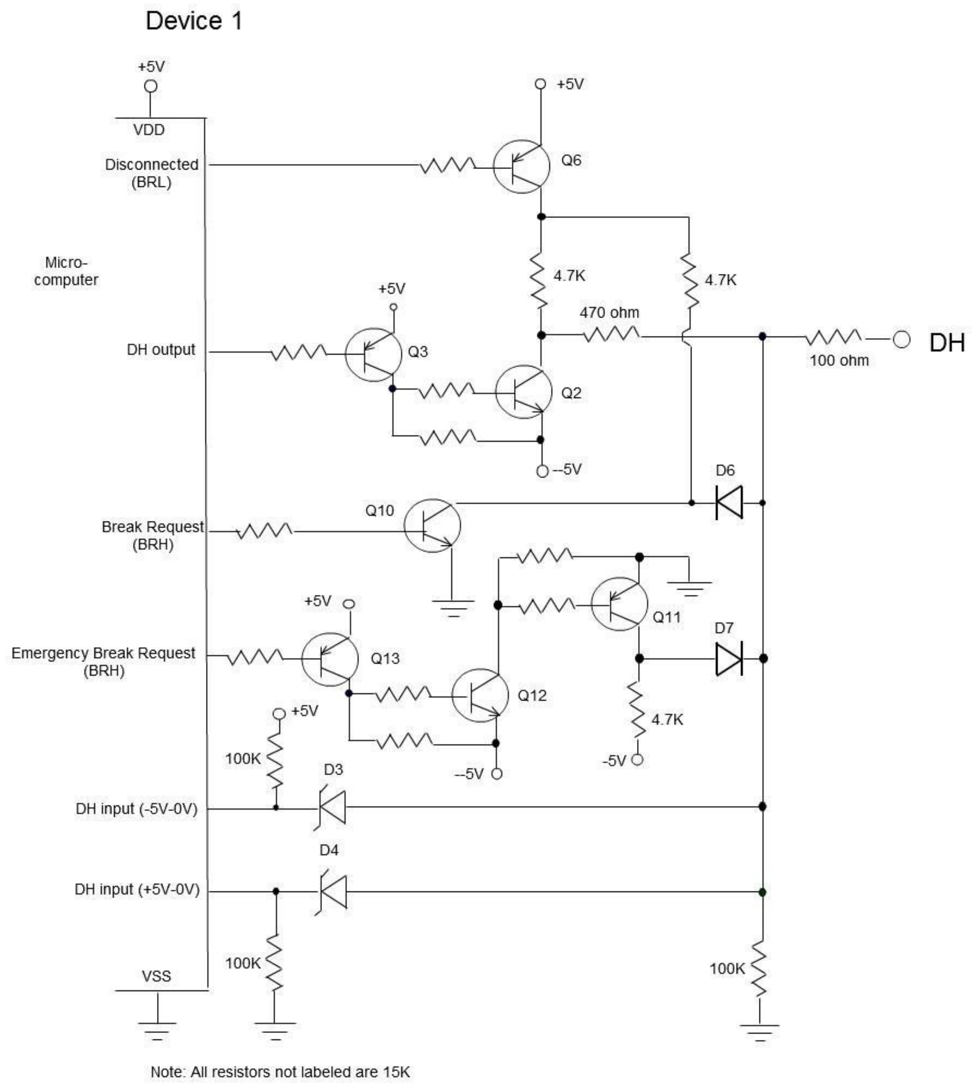
Device 1



Note: All resistors not labeled are 15K

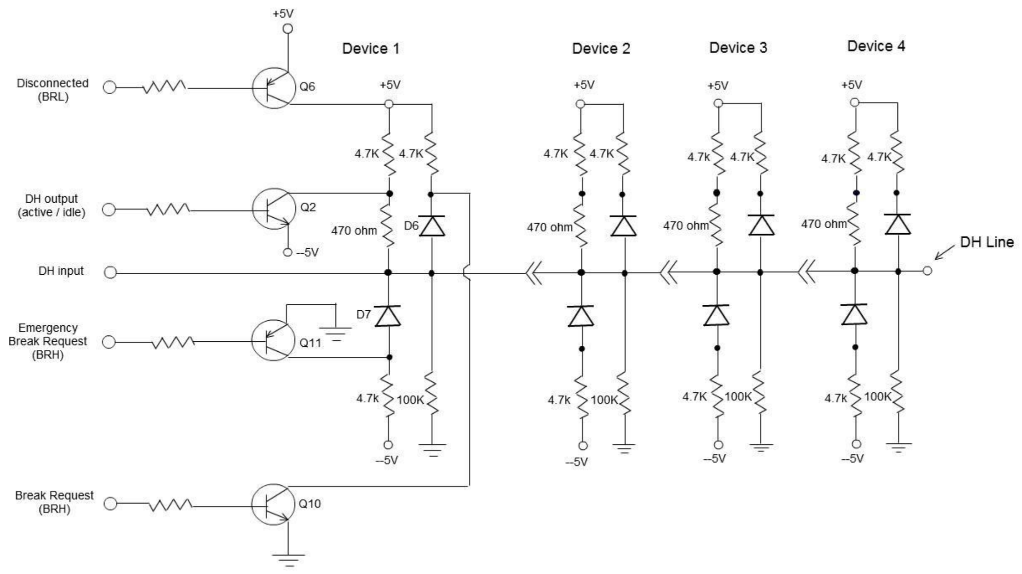
In the latest development, this break request-high signal will also be able to override the active state voltage on a data line (this addition is the “emergency break request” feature). The break request signal would then have the highest priority of the four data line states, and any device can use this signal at any time to take immediate control of the data line. This new state would need a second switching transistor to bring the opposite voltage level (active state) to ground potential since a switching transistor can switch current on and off in one direction only. Therefore, the ordering of four states from strongest to weakest would become (1) break request-high, (2) active state, (3) idle state, and (4) break request-low.

The diagram below modifies the previous circuit to include this new “emergency break request” feature. Q11 has been added which is the second switching transistor for break request. There are also two more level shifting transistors Q12 and Q13. D6 and D7 provide circuit isolation since the current is one-way for each switching transistor. A 470 ohm resistor has also been added for limiting the active state current. I have shown only the circuitry for the DH line, as the component count is getting a little high (14 transistors and 8 diodes total). The circuitry for the DL line would be in a similar format as shown here anyway, except it would be in reverse “mirror image” format. This diagram is pretty much what the finalized version of an SSB circuit would look like.



This circuit is actually simplified from what it was in an earlier development. I had designed a special ‘wired AND’ circuit which was used for receiver responses. It made sure every receiving device responded to a signal. But this function is now taken care of by the Disconnected (BRL) signal.

The following diagram is basically the same circuit as above, but I removed all of the “level shifting” components to simplify it so the reader can get a better understanding of how the circuit actually works. Only components that directly affect the data line are shown. This schematic also shows how the circuit will look when connected to other devices.



To make the diagram simpler, I left out the four driver transistors on Devices 2, 3, and 4, but the reader can assume that they are still there according to what is shown on Device 1.

The four possible states for the two data lines which are each designed for special purposes according their designated voltage levels are summarized as follows:

+5 volts	-	idle state for DH; active state for DL
0 volts	-	('strong' ground) break request-high (BRH)
0 volts	-	('weak' ground) missing connection or break request-low (BRL)
-5 volts	-	idle state for DL; active state for DH

3. Fault Conditions

The reason that the higher order break request was added is because the original one can only be used to interrupt transmissions if at least one of the two data lines is in the idle state. If for some reason a fault occurs where both data lines are tied up in the active state voltage potential, it would be necessary for this condition to be interrupted, and then have each device (or at least the device at fault) brought into a shut-down or low-power mode of operation. This is the reason for the 'emergency break request', which also is a part of break request-high (BRH), but uses the second switching transistor. As it is applied to a data line, it will switch any active voltage on the line to ground directly through this switching transistor. There still is a possibility that even an emergency break request will fail to bring the device at fault out of its active state condition. In this case all devices on the bus line would probably have to go into their shut-down or low-power mode. This 'emergency break request' will be very rarely used, if at all, and almost all break requests will be issued by the BRH signal that controls the first switching transistor, which pulls the idle state to ground. However, this change alters the ordering of states from strongest to weakest, so that now break request-high is the strongest of the data line states, where before the 'active' state was the strongest.

If a fault or disconnection is found anywhere in this circuit, it would generally be detected by the devices connected to the data lines and the devices with the fault condition would shut down to a low-power mode instantly and possibly record or display a message indicating the problem. For example, if the one of the connector pins had an open circuit, both devices on a two-device SSB configuration would note this and shut down accordingly.

One potential problem with using 'zero' state in the 'weak' ground mode is when the connector is connected and one or more devices are in their operating mode. In order for the data line to be at zero volts, both ends would have to be in their high-impedance 'zero state' mode. If any one of the devices were in an active state or idle state on this line, the entire line would be set at +5 volts or –5 volts. It would be impossible for one device to signal a 'break request' to another device using this 'weak' ground because the +5 volts or –5 volts from the opposing device would override this. Therefore, a 'strong' ground was designed which would be used for the break request, and the 'weak' ground for the other conditions, such as connected/disconnected, device on bus line with power off, or some other fault condition.

Another potential problem is that the circuit as shown would not be able to detect an open circuit condition between one of the resistors and ground unless the connector is disconnected and/or special circuitry for this is added. When the 100K resistors on both sides are connected, the data lines would see a 50K resistance to ground (two 100K resistors in parallel). If one of the resistors inadvertently becomes disconnected, the corresponding data lines would see a 100K resistor to ground. This couldn’t be detected with digital logic circuitry and an analog comparator would be needed in this case.

4. Sensing Connection/Disconnection

When two or more devices are connected together in a simple common bus line arrangement, the two data lines will normally be in their idle states, and would go into their active states momentarily as pulses are applied from the transmitter on one line and the receiver on the opposite line. With this kind of system, it becomes apparent that there is no reliable way for the hardware to determine if a device's connector is plugged in or not.

The only effective way that can be found to determine if there is no connection is by momentarily turning off the drivers through software which sets the high impedance-input state and checks to see if the data line is still pulled low (or high) by the other side. The device could be put in a check mode for 1 millisecond for every 1 second period, for example. If the other device (or devices) are turned off or there is no connection, the line would be at zero volts through the 100K resistor. In this case a shutdown occurs, and the disconnected device would go into a low-power mode where all of its data lines are held in their high-impedance state and kept at zero volts through a 100K resistor. Another important consideration is that the DH and DL pins should not be left floating in the low-power mode, but they should be set to an acceptable voltage level through the 100K resistors.

Once there is a disconnection and a device is in the low-power mode, it should be possible to determine when the connector is plugged in again and thus bring the device back into its operating mode. When the device is plugged in again to a network configuration, it will sense the voltage levels on the data lines immediately and power up accordingly. But if there is a two-device configuration, and both devices are inactive with their inputs held in a high-impedance input state, they would not be able to tell the difference between a connection and a disconnection. Each of the four inputs would only see a ground potential (zero volts) through either a 100K resistance (no connection) or a 50K resistance (connected). As I mentioned earlier, an analog comparator circuit would be needed to detect this.

A simpler way would be to apply a 1 millisecond pulse for every 1 second period in a similar manner as when a check is made for disconnection. This time, instead of momentarily putting one data line in the high impedance-input state, the data line would be momentarily taken out of the impedance-input state. An input on one end would become and output for 1 millisecond in its idle state voltage. The high-impedance input on the other end would immediately detect this change and would assume that the connection is made and go back into normal operating mode. The end that sent the pulse would then detect the power up on the opposite end also power up into its active state. All inputs would then become as outputs with idle state voltages.

There is one main difference between this pulse sending check for connection and the one for disconnection. The check for disconnection requires all four I/O pins to send a pulse at certain intervals. This is done so a thorough check can be made for an open circuit, the device opposite end being turned off, etc. The check for connection only requires one I/O pin on one end to send a pulse at certain intervals (All four I/O pins could be sending pulses, but this is not necessary). As soon as a positive response is found on one pulse (or several pulses, to add a ‘debounce’ mechanism), the entire circuit on both ends would become activated accordingly.

5. Data Transmission and Reception

So far I have only discussed idle state circuit conditions when there is no actual data being transmitted from one end to the other. I have done this to verify the basic structure of the circuit and the extent of the hardware needed for reliable data transmission to make it relatively error-free with adequate fault detection. The following discussion will move away from the hardware aspects of SSB and consider more in the software end of the spectrum as the sequential timing and logic for data transmission and reception is explained.

It can be generally assumed that the Simple Serial Bus would be a peer-to-peer half-duplex link between two devices. There should be no specific requirements placed on the serving devices, and the data transfer speed should be self-adjusting and entirely dependent on the instantaneous conditions over the link. The auto-adjusting features of SSB will allow it to accommodate devices of different speeds. By using these features, along with a special signal ‘debounce’ filter algorithm, SSB should perform very well in noisy environments.

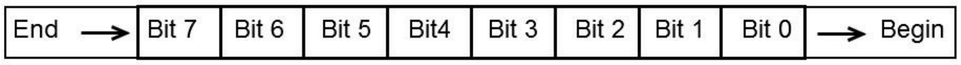
Another major difference between SSB and other technologies is that SSB utilizes a pseudo-differential transmission channel that differs both from single channel and normal differential lines. Two data lines are required as in the normal differential lines, yet they can be controlled individually and independently of each other as though they were single channel links.

As a device transmits or receives data, it does so one bit at a time, so that the data transfer can be thought of as a sequence of bits rather than bytes. Each sequence of bits can be thought of as a block of data, and each block
is terminated with a ‘break request’ from either the receiver or transmitter which signals the end of the block. This eliminates a lot of extra overhead on the system and also allows the transmission to be interrupted during any bit in the sequence by a ‘break request’ from either the receiving end or the transmitting end.

The process of data transmission in SSB starts when the transmitting device issues a ‘beak request’ followed by a 6 bit address which is for the intended receiving device. This 6 bit address is a recent development for SSB networking, and is used to verify that there is a receiving unit connected to the data lines and is enabled to receive data. More will be said about this later, but for now it will just be stated that the break request-low signal (BRL) will be used to confirm the receiver’s presence on the data lines. Since this is a ‘wired AND’ signal, all non-transmitting devices will issue this signal except for the intended receiver. So if the data lines are both not at ground (BRL) potential, then the transmitter has its confirmation, and will commence with the transmission. If any data line is at ground (BRL) potential, then the transmitter assumes that the intended receiver is not available, and the transmission is aborted, and the transmitter issues another break request-high (BRH) to signal to the remaining devices that the transmission has ended.

If the transmitter has received the confirmation, it will immediately start transmitting one bit after another until the transmission is complete for a block of data or a 'break request' is issued by either transmitter or receiver for some reason which terminates the transmission during the transfer of the interrupted bit in the sequence.

Since this transmission process is bit oriented instead of byte oriented as in other protocols, a sequence of eleven bits for example would be transferred as a sequence of eleven bits instead of two bytes (sixteen bits) as in other protocols. If the data length is more than one bit, the least significant bit is always transmitted first, so that an entire byte of information would be transmitted as:



If the data length is more than one byte, it would be transferred as one continuous stream of bits unless the 'break request' feature is used for one device to signal to the other that that an entire byte has been transmitted (or received). The transmitted data sequence doesn't have to be broken down into bytes, but it could be a block of data bits of any reasonable length. It would be possible to transmit very long data sequences using this method with no interruptions, but in order to avoid errors, it is recommended to break the long sequence down into several blocks of data.

If there was a data transfer done using blocks rather than one continuous sequence, there should be an 'agreement' between transmitting and receiving units as to how long each block of data bits should be. For example, if the transmitter plans on sending several blocks of data with fifteen bits each, the receiver could know this block size number (15), so that it can issue a 'break request' every time a block of fifteen bits has been received

If there is no agreement between transmitter and receiver about the number of bits in each block, the transmitter can still send several blocks with fifteen bits each, but they would be counted as separate data transmission sequences, and the transmitter would send a ‘break request’ signal to the receiver whenever the transmission of a data sequence is finished.

The ‘break request’ signal doesn’t have to be sent only at the end of the data sequence, but can be sent anytime during the transmission of any bit by either transmitter or receiver. This interruption can occur for many reasons, such as data buffer full, error in data transfer, or another higher priority task is to be performed. Therefore, the data transfer process will continue until the transmitter or receiver issues a final break request signaling to the other device that all of the data has been transmitted or until one device signals to the other that the transmission should be stopped, and the remainder of the data transfer process would be aborted. If the receiver issued a break request to signal a condition such as error found in transmitting or receiving data, this would cause the transmitter to resend the data previously sent. Sometimes the internal capabilities of a receiving device only allow a limited number of bits to be transferred, so in this case the receiver would issue a ‘break request’ on the last bit allowable, and the transmitter would cease immediately and abort the remainder of the data block transfer, if any. Up to this point, a break request has always indicated the start or end of data transfer sequences, where the transmitting device issues the initial break request, and either transmitter or receiver can issue the data block ending break request.

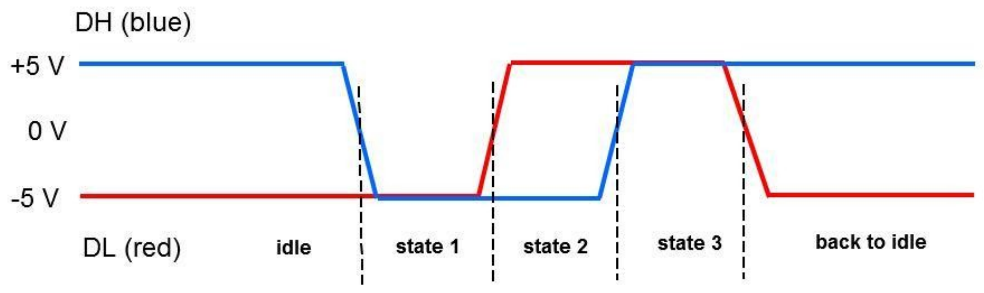
The timing diagrams for a ‘break request’ issued by a transmitter differs in format from a ‘break request’ issued by a receiver. This will be explained later following the data transfer timing diagrams.

6. Timing Diagrams

The timing diagrams shown below will demonstrate how the actual transmission of a bit of data takes place on the two data lines (DL and DH). Voltages in the range of +5 volts to –5 volts are shown in the vertical coordinate, and the time between the various states are shown in the horizontal coordinate. The DH data line is indicated by the blue timing waveform, and the DL data line is indicated by the red timing waveform.

Each of the D-lines are used specifically for one the two binary data bit values. The DH data line is used basically for transmitting and receiving a logic ‘1’ value, while the DL line is for transmitting and receiving a logic ‘0’ value. During the transmission of one of these two values, the opposite data line automatically becomes a data acknowledgement signal line where the receiver would provide confirmation to the transmitter that a valid bit of data has been received.

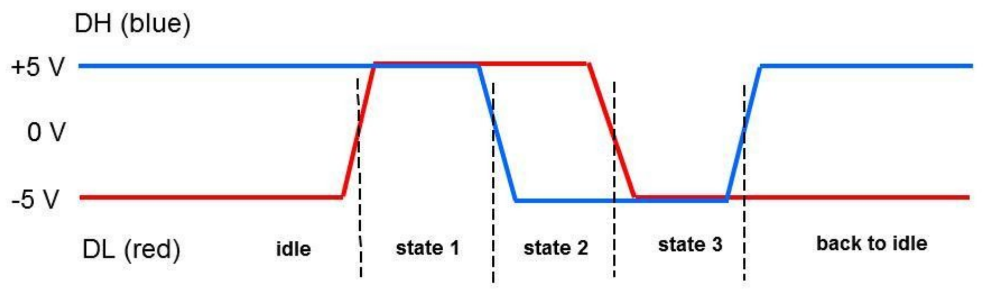
An example diagram showing the process of transmitting and receiving a single data bit is shown below for the transmission of the logic ’1’ value. The DH and DL lines start in their idle state voltages as explained before, and then the transmitter initializes the process by energizing the DH line from its idle state (+5V) to its active state (-5V). The line is kept in this state and held until the opposite line (DL) is activated by the receiver as confirmation (state 1). When the transmitter receives this acknowledgement signal, it can assume that data transfer was successful, and proceeds to deactivate the DH line back to its idle state (state 2). During this time, the receiver was waiting for the DH line to return to idle as it indicates to the receiver that the transmitting device has received the acknowledgement signal. After sensing this signal, the receiver will be able to return the DL line back to its idle state which completes the data transfer process for a single data bit value of ‘1’ (state 3).



The data transfer process for a single bit of logic value ‘1’ data can be summarized as follows:

State 1: Transmitter puts DH to active state (-5V)
State 2: Receiver puts DL to active stage (+5V)
State 3: Transmitter returns DH to idle state (+5V)
State 4: Receiver returns DL to idle state (-5V)

The logic ‘0’ value is sent in a similar way but in a kind of ‘mirror image’ format. Again, both lines start in their idle states as before, and then the transmitter starts the process by changing the DL line from its idle state (-5V) to its active state (+5V). The line is kept active and the transmitter waits for the opposite line (DH) to be activated by the receiver as confirmation (state 1). As the transmitter receives this acknowledgement signal, it means that the data transfer was successful, and then returns the DL line back to its idle state (state 2). Finally, the receiver returns DH back to the idle state as it receives confirmation that the transmitter has received its acknowledgement signal (state 3). This completes the data transfer process for a single data bit value of ‘0’.



The data transfer process for a single bit of logic value ‘0’ data can be summarized as follows:

State 1:	Transmitter puts DL to active state (+5V)
State 2:	Receiver puts DH to active stage (-5V)
State 3:	Transmitter returns DL to idle state (-5V)
State 4:	Receiver returns DH to idle state (+5V)

The transmitter was also waiting for the final receiver signal in state 3 of both diagrams and both lines are back in their idle state, where a new bit transfer can be initiated or the control could be transferred somewhere else depending on interrupts, break conditions, etc. Of course, a certain “debounce” time must pass before another event can take place.
This data transfer method should work well even when one of the devices operates at a clock speed far greater than the clock speed of the other device. There is no clock synchronization between devices, so the transfer speed is self-adjusting in the sense that one device always waits for the opposite device to send its acknowledgement signal.

There is no time limit for keeping a line in an active or idle state in any of the three states shown in the above diagrams. Any of the two lines can be kept in any state indefinitely while waiting for the other device to respond. This would make this SSB protocol completely independent of device speed.

As seen above, this form of data transfer ‘handshaking’ reverses the functionality of the two data lines in the transfer process of logic value ‘1’ and logic value ‘0’. The DH line is a data transmission line and DL is the receiver acknowledgement line for logic value ‘1’. The DL line then becomes the the data transmission line and DH becomes the receiver acknowledgement line for logic value ‘0’. They would also operate at opposite voltage levels for these bit transfers, so in a sense, the pattern for the logic ‘1’ transfer and the pattern for the logic ‘0’ transfer are mirror images of one another.

This feature of using opposite voltage levels on the two data lines provides an error detecting feature that would prevent errors in mixing up the two data lines when connecting a new device. The DH line has an idle state voltage of +5 volts and is usually at this level at all times while connected unless a data transfer is taking place where there are short pulses to –5 volts. In a similar way, the DL line has an idle state voltage of -5 volts except when there are short pulses to +5 volts. Checking these idle state voltages can easily determine which data line is DH and which one is DL.

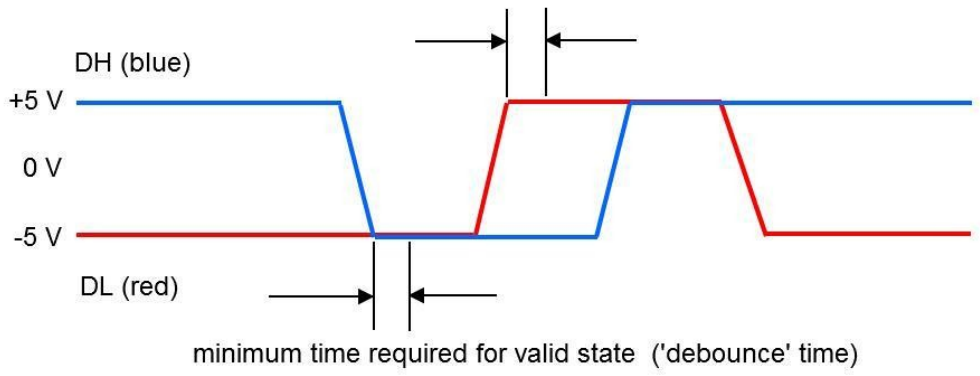
If the DH and DL did not operate at opposite voltage levels, but instead operated at the same level as other communication protocols normally do, the pulses for data bits would appear the same way for both logic values. The acknowledgement pulses would also appear the same way and the two data lines could easily become reversed. A logic value ‘0’ could be mistaken for a ‘1’, and a ‘1’ could be mistaken for a ‘0’.

This data transfer method would eliminate at least one and possibly two the the handshaking control lines used in other serial data transmission protocols. Most of these protocols use a ‘Data Valid’ signal line along with a separate data bit line as the transmitter sends its data. The receiver would then respond with a ’Data Accepted’ signal on a separate control line in the same fashion as is done here with the opposite data line. This new SSB method combines the ‘Data Valid’ signal with the data bit signal into just one signal line as their timing and polarity is the same in each case for both logic ‘0’ and logic ‘1’ transmission. The opposite line always becomes a control or ‘confirmation’ line when transmitting data.

Other control lines, such as DSR (Data Set Ready), and CTS (Clear to Send) normally found in protocols such as RS-232 can be effectively eliminated since these signals basically just let the transmitter (master) know that the receiver (slave) is ready to receive data. With this SSB protocol, these signals are automatically implied when both data lines are in their idle states and a certain ‘debounce’ time has passed since the last data line transition. Once a transmitter takes over the data lines and initiates a transmission, only the intended receiver is automatically enabled to receive data while the other receivers on the bus lines (if any) are disabled.

The ‘debounce’ time is a certain minimum time that must pass after the final transition in state 3 where both data lines are in their idle states before a new data bit can be sent. In fact, any time there is a transition at all in the previous timing diagrams, there must be a certain ‘debounce’ time where the data line is required to be in the same state for at least this minimum time in order to be considered valid by the receiving device. This ‘debounce’ time can be programmed individually for each device, and each device would use this same debounce time for detection of ‘idle’ state, ‘active’ state, ‘break request’, ‘disconnected’, and data bit transfer complete.

The SSB configuration does not use edge-sensitive logic like some protocols do, but uses level-sensitive logic instead. The diagram below shows what a typical ‘debounce’ time period might look like for detecting an ‘active’ state.

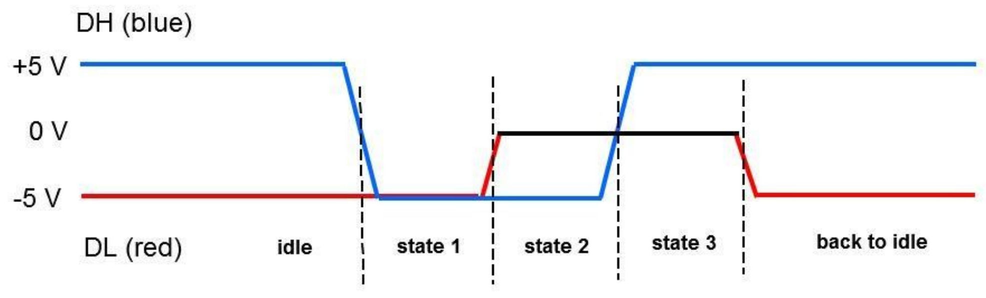


7. Break Request

I have explained earlier about how a ‘break request’ signal is used anytime by either transmitter or receiver to indicate end of data transfer, unable to receive data, error in data sent, and many other conditions. This is a multi-purpose feature that would allow this transmission protocol to eliminate many of the control lines found in other serial communications methods, such as DTR, DSR, RTS, etc. There would also be no need for certain software methods, such as an enumeration process, as the information needed about the device is already assumed through DIP switch settings or other similar parameter setting methods.

The following diagrams will show how a ‘break request’ issued by either transmitter or receiver can abruptly halt any data transferring activity, and then have the data lines set back in their ‘idle’ mode. The break request timing diagram format differs between transmitter and receiver as follows: When the receiver issues a break request, instead of acknowledging the data bit sent by activating the opposite data line, it places it at ground potential, or zero volts (state 1) by activating BRH for that line. At this point, the transmitter is notified that the receiver cannot accept more data bits and terminates the transfer by returning its data line back to its idle state (state 2). This confirms that the data transfer has been terminated, and marks the end of the data block. After detecting this transition, the receiver is also due to reset the opposite data line to its idle state (state 3).

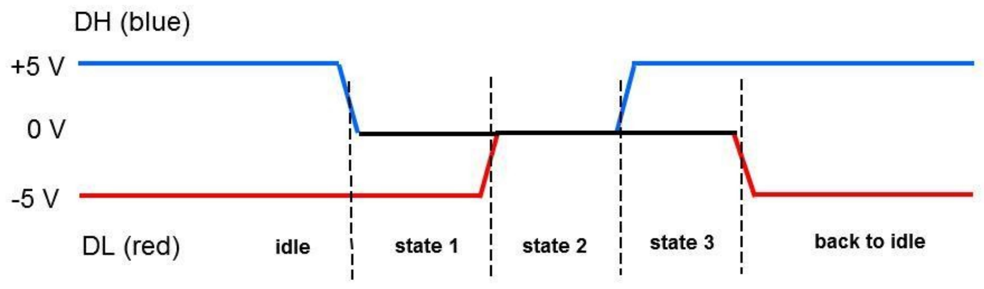
At this point, both data lines are in their idle states, and both devices have been given information that data transmission has stopped, and the data lines are in their normal operating mode again, waiting for the the transmission of a new block of data to begin. As shown below, the timing diagram for the receiver issuing a break request is the same as the one for a normal data bit being sent, except that the acknowledgement pulse from the receiver is ‘clipped’ to zero volts.



state 2 = break request from receiver

Under normal conditions, the break request will only need to be issued by the transmitter to mark the beginning and ending of a data block transfer. (Although a transmitter can interrupt data transfer for other reasons too, such as error conditions.) As shown in the diagram, the transmitter starts by placing one of the two data lines (it doesn’t matter which one, but not both) at ground potential (zero volts) by activating BRH for that line. The transmitter then waits for a response as the receiver confirms the break request signal by setting the opposite data line to ground potential (state 1) by activating its BRH signal. At this point, both data lines are at the zero volt level indicating break request. As soon as transmitter detects that its break request has been verified, it will return its data line back to the idle state (state 2). This transition would also indicate to the receiver that the break request has been verified and it will reset the opposite data line to its idle state (state 3).

So at the final stage, both data lines are back in their idle states, and both devices know that data transmission has stopped, and the data lines are in their normal operating mode again waiting for new transmission block. As shown below, the timing diagram for the transmitter issuing a break request is the same as the one for a normal data bit being sent, except that both the data bit pulse from the transmitter and the acknowledgement pulse from the receiver have been ‘clipped’ to zero volts.



state 1 = break request from transmitter
state 2 = confirmation from receiver

A break request pulse issued in this way can be considered as a third pulse type along with the first two pulse types for logic value ‘0’ and logic value ‘1’. So instead of being a binary protocol, the system would have essentially a ‘trinary’ protocol. A sequence of two bits would have 3 * 3 or 9 possible combinations instead of the 2 * 2 or 4 combinations possible in a binary code. This would add much control capability to the system as each bit sent can contain not only data bit value information, but also control information. This is how the break request feature can replace some of the control lines found in protocols like RS-232, such as RTS, DTR, and DSR.

The break request signal should be such that it dominates, or has priority over the active and idle state signals on each of the two data lines. Therefore, it would be possible for any device to issue a break request at any time on either bus line, and it would take control over the data lines and abruptly stop any transmissions taking place. This would allow a higher priority activity to be done on the bus lines, such as an emergency procedure. For example, one device could be drawing excess current, and the data lines need to be brought to ground level (0 volts) immediately.

The order of precedence, from strongest to weakest of each of the possible states on the common data bus lines are as follows: (1) break request, (2) active state, (3) idle state, and (4) disconnect ground. I have thoroughly discussed the first three, and I will mention more about the disconnect ground later.

8. SSB Networking

So far, this document has mostly described the basics for SSB communications between two devices, so the break request as described was only used to mark the beginning or ending of a data block or some other condition which requires data transmission to stop. But when more than two devices are connected, as they would be in a network configuration with common bus lines, the break request signal could be used for other purposes.

When a transmitter sends data in this configuration, it isn’t necessary for all of the non-transmitting devices to receive data, but only the intended receiver should receive it. In this case, the transmitter needs to send a message indicating the address of the intended receiver, so each of the non-transmitting devices can distinguish this, and enable or disable themselves accordingly.

This addressing message would be sent by the transmitter just before the data message itself, and this would be the SSB protocol for common bus line networking. The addressing message itself would be a simple binary number about 6 or 7 bit long, which can address up to 127 devices. This binary address would mark the beginning of a sequence or block of data to be transmitted.

This is a simplified approach to adding additional devices to the basic two-device SSB communication system. One of the main problems with this approach would be the possibility of transmission collisions. One of the ways of solving this would be by using the ‘random delay’ concept similar to what is used in Ethernet. A transmission collision can be detected by all devices on the bus line within the first few bits of information, which can be used as a signal to shut down all transmissions for a certain time period. This time period is programmed to be a ‘random delay’ which would be different for each device to avoid another collision when data transmission starts again. If the two devices that had their transmission cancelled are still actively waiting to re-transmit, the device with the shortest delay period would have an opportunity to transmit again with almost no chance of collision.

Another major problem that needs to be solved with SSB networking is how to determine how devices are to be notified when any other device (or devices) are plugged in or unplugged from the common bus line. It would be a simple matter to determine when devices are removed (exclusion). When another device tries to send an addressing message to it, it would immediately sense no response, and thus determine that the device is excluded. Since each device on the bus lines are listening to the transmit and receive activity, all non-transmitting device would also be able to sense device responses or no responses and they would be able to record information for the removal of a device accordingly. The removed device itself (if it's still on) would use the pulsing technique described above for detecting whether or not its connector is plugged in or not, and it will disable itself by going into the low-power 'zero' state mode (BRL) immediately.

Detecting the inclusion of a device is a little more difficult to solve. In a normal network mode, transmitters would be sending data only to those devices with ID numbers that have been recorded for the network at that particular moment. Each device's ID number would correspond to a bit in a bit-map. If a new device plugs in with an address that is different from these recorded addresses, it would be ignored since its address is not included with those on the bit-map (each device has exactly the same bit–map set up for connected devices).

The most recent development in SSB networking that will solve both the collision problem and the inclusion/exclusion problem is a scheme that uses a rotational "taking turns" concept. With this algorithm, only one device can have control of the bus lines at any given moment. Each device gets a time slot for this as this bus line control is passed sequentially in a rotational pattern to each connected device, one at a time. As one device takes its turn for controlling the bus lines, it will have the option of transmitting a block of data. If it has no data to transmit, it just passes control of the bus to the device with the next higher ID number. If this number is already the highest, then it passes to the lowest. If the bus-controlling device has data to transmit, then it will transmit the data sequentially as explained above, and then transfer control to the next device.

Each device looks up the address for the next corresponding device in the bit-map as it transfers control to it. So each connected device is addressed directly from the preceding one. This scheme solves the collision problem since it is only possible for one device to transmit at any given moment. The problem of exclusion is also easily solved as explained above when there is no device response to an address. However, this approach still doesn't solve the inclusion problem since the addressing messages are sent only to those devices that have corresponding bits set in the bit-map. If a new device is plugged in, it will be ignored as before.

There is a way to solve this using the same basic rotational "taking turns" method. Instead of addressing each connected device directly according to a bit-map, the devices could be addressed in a sequential manner with the next ID number (ID+1). As one device passes its bus line control over to the next device, it would start by incrementing its own ID by one and send an address message using that ID. If there is no device response for ID+1, it will send the message to ID+2 and so on until a valid response is received and bus control would then pass on to another device. This process would continue for one complete revolution. In this way, a complete scan is made of all possible addresses, and newly added devices (inclusion) can easily be determined and plotted on the bit-map.

This ID+1 version of the rotational concept would not need to done for every cycle, but instead it would be done for one cycle for every 63 cycles of the direct addressing form of the rotational concept. This would combine the two concepts to create an efficient networking scheme with minimum overhead. The detection of devices added and devices removed from the bus lines is considered low priority compared to scanning connected devices for data to transmit.

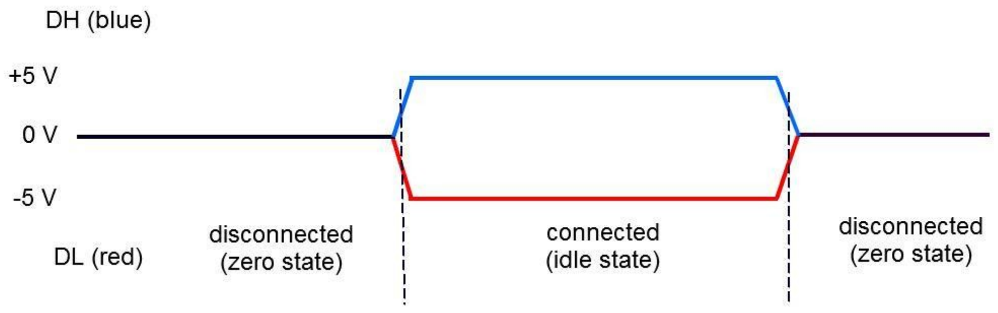
So the only difference between SSB networking and regular two-device SSB communication is the inclusion of the intended receiver address message just before the data message and the rotational “taking turns” concept that allows only one device to use the bus at a time in sequential order. The collision problem and exclusion problem are solved using this method. The inclusion problem is solved by using the ID+1 form of addressing once every 64 cycles. The direct form of addressing using bit-maps is used for the other 63 cycles for higher speed and efficiency.

9. Disconnect Feature

There is a disconnect when there is no connection between the disconnected device and any other device. A device can also be considered disconnected if one of its data lines has a bad connection or there is some other fault in the device's driver circuitry. If one of these conditions occurs, the device automatically turns all of its output driver transistors off, and the device goes into a low power mode with high-impedance data line inputs. These inputs are also connected through a high value resistor to ground, so the device's two data lines would be at ground potential (0 volts), assuming that both lines are disconnected from any other device.

This would be a 'weak' ground, and it would be the weakest state of all the four data line states, and any connected device can dominate a data line in this 'weak' ground state by applying an idle or active state voltage. The only way to bring that data line back to ground potential (0 volts) then would be to override the idle or active state voltage with a break request. This is a 'strong' ground and is the strongest of the four data line states.

The diagram below shows how the data line voltages would look as an individual device is connected and later disconnected from the other device (or devices). If the devices were set up according to the regular two-device SSB communication format, both devices would show this waveform on their data lines. If the devices were set up for SSB networking, only the disconnected device would show this waveform, while the data line voltages on the networked devices remain the same, since they don't sense a valid disconnection. They would only sense the exclusion of one of the connected devices.



It may also be desirable to use the ‘disconnected’ state when it is known that a device will not need to transmit or receive data for a long period of time. A device in this case will be able to put itself in the ‘disconnected’ state even though it is connected to another device (or devices). The device would be in a very low power mode where all of its driver transistors are off, and this would become a third state as found in typical tri-state circuitry where the inputs are high–impedance and are connected through a high value resistor to ground. This method would increase noise immunity and network efficiency.

10. Connector Specifications

The following chart is a preliminary electrical specification for the basic SSB circuit as described. This shows the voltage ranges each or the four data line states (active, idle, ‘strong’ ground, and ‘weak’ ground). The transition between states is done with hysteresis.

	Idle	**Active**
DH	+2.4V to +5V	-2.4V to -5V
DL	-2.4V to -5V	+2.4V to +5V
Zero	-0.8V to +0.8V	
Power	+4.5V to +5.5V DC 1A max.	

The basic SSB circuit will use a 4-pin connector with pin assignments for VCC, DL, DH, and GND. The fourth pin can be considered optional as it is used only to transfer electrical power (VCC) between devices. This would work well for configurations where only one device on the network has an internal power supply, but if more than one device on the network has a power supply, or to shut down power transfer when there is excessive current being drawn, a power distribution circuit should be used as discussed in the next section.

Therefore, the connector would be either 3-pin or 4-pin, depending on the voltage supply configuration. The connector should be designed such that it mechanically prevents itself from being plugged in a reversed state. If possible, a connector should be found which would be able to drive both the data lines to ground directly or through a small value resistive load in the disconnected state.

Also, the recommended connector should be able to connect the GND pin before the data line pins. If the VCC pin is used, it should be connected before the data line pins also.

11. SSB Power Distribution

There are times when devices hooked up to a SSB circuit don't have their own internal power supply, and it becomes necessary to rely on the power supply in another connected device for operation. In this case a fourth pin on the connector is needed which will allow power to be transferred between devices.

This creates a variety of problems with a common bus line configuration, however. The first problem is when there is more than one device hooked up which have internal power supplies. This question would be how to determine which of these devices would supply power to the non-powered devices on the network (if any). The next consideration would be how to design the power transfer circuit so that each of the devices with internal power can operate normally with no interference from other powered devices in the network.

The second main problem is on how to handle a situation when there is excess power being drawn by one of the devices in the network. In this case, the fourth (power transfer) pin should not be used at all, and effectively removed from the circuit, so that a 3-pin configuration is used. This condition also shuts down all non-powered devices in the network, so that only the self-powered devices remain in operation. As soon as the defective device is disconnected from the network, each of the connected devices would sense this, and power themselves up back to normal operation.

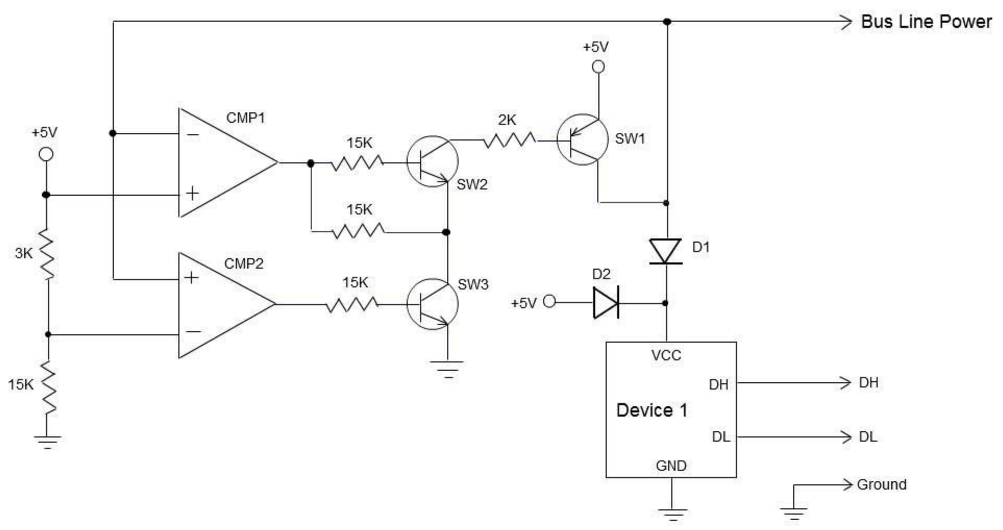
A standard USB network uses the "tiered star topology" to solve these problems on how to distribute power between devices. This arrangement uses a hub system where each hub on a device can supply power to one and only one device in the downstream direction. A downstream device cannot transfer power to an upstream device (host). The enumeration process informs the upstream device what the power requirements are in the connected downstream device. In this way, there is no confusion about which devices will be transferring power to another, and each of the self-powered devices can use their own power as designated through the enumeration process.

The problem of current overload is easily solved with this method, since each downstream device connects to only one upstream device. The upstream device can just shut down power transfer to the one device below it that has is drawing excess current. And each or the remaining devices can continue with normal operation, whether it's self-powered or not.

The SSB network will not use many of these advanced features, so its power transfer capability will have to be somewhat limited. With a common bus line approach, there can only be one device that can be selected that will transfer power to the non-powered devices connected to the bus. This device will be selected according to the voltage level of its internal power supply. In normal operation, there are slight differences in these voltage levels between devices, and the one that is found to have the highest voltage level automatically selected as the main network power source. Since there is some hysteresis using this method, any self-powered device on the network can continue using its own internal power supply without depending on the main network power. However, any non-powered device will have to depend on the power received from the device selected as the main power source.

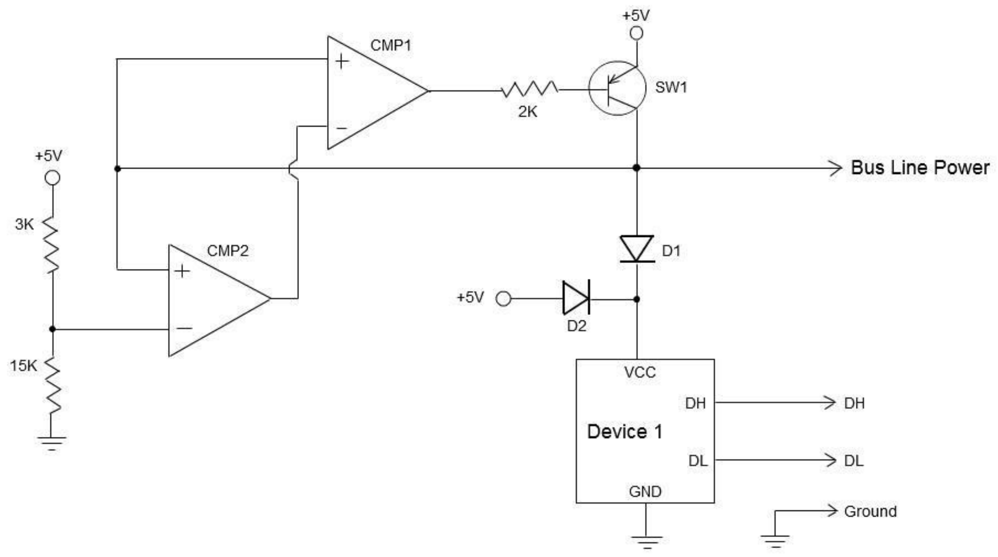
The device with the highest voltage level on the network is selected according to the circuit below, where CMP1 compares the bus line power voltage directly with the voltage of its own internal power supply. If the bus line voltage is higher, SW1 is turned off, and the bus power line is used to supply power for the device. If its own internal supply voltage is higher, SW1 is turned on, and the device will become the main network power source, unless there is a current overload somewhere, in which case CMP2 will turn SW1 off, and the device will again rely on its own internal power supply, if any. If there is a current overload condition on the network, all devices on the network will detect this, and this effectively disconnects the bus line power pin on the connector for each device. The self-powered devices can still operate normally on its own power supply through D2. However, each of the non-powered devices will have to be shut down, which is one disadvantage over the USB protocol. As I stated before, normal operation resumes as soon as the device at fault is removed from the network.

The circuit below describes the concept as discussed so far. CMP1 compares the bus line voltage with the internal supply voltage, and CM2 compares the bus line voltage to a lower level based on current overload conditions. SW2 and SW3 form a “wired AND” circuit, so that SW1 is on only when the bus line voltage is greater than the overload voltage and less than the internal supply voltage.



CM2 has priority over CMP1, so that if CMP2 shows excess voltage drop, CMP1 has no effect on circuit operation. The circuit would still have to apply short pulses to the power line periodically to check if the device at fault has been removed. If it is, then the network can go back to normal operation. The diodes are important because they prevent a higher level voltage source from shorting out to a lower level voltage source on the bus line. If bus line voltage is greater than the internal power supply voltage, D2 does the current blocking. If the internal power supply voltage is greater than the bus line voltage, D1 does the current blocking.

This circuit also works if there is no internal power supply preset, as this would register as zero volts on the comparators. SW3 would be on, and SW2 would be off, so then SW1 is off, which disconnects the internal power and allows externally supplied power to be used. Another circuit, which has these it features, but may be a little simpler, is shown as follows:



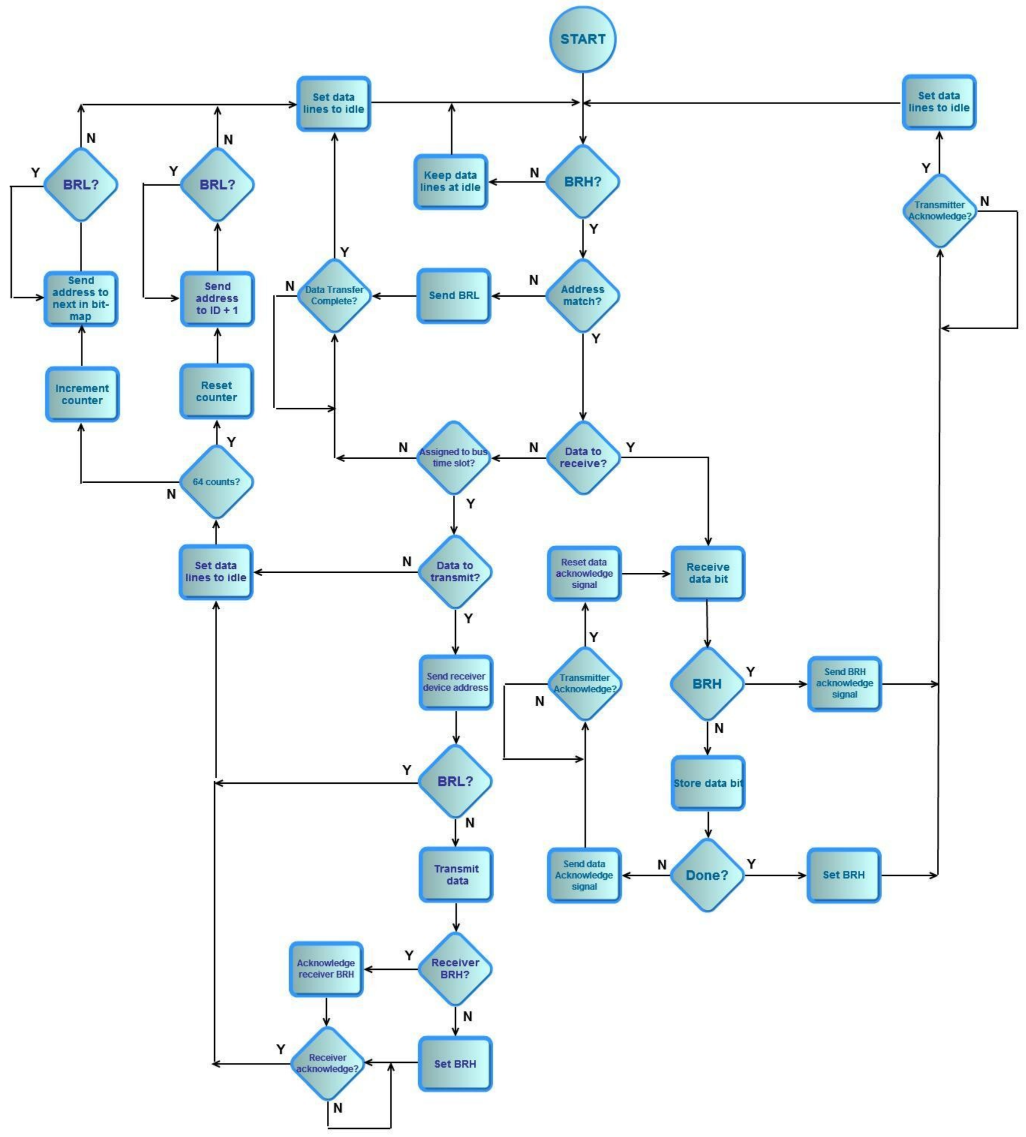
In this circuit, the output of CMP2 is tied directly to the negative terminal of CMP1, and eliminates the need for the “wired AND” transistors. However, this doesn’t provide a reliable reference voltage, as the bus line voltage should be compared directly with the internal (+5 volt) supply voltage. There is also a problem when the bus power line is shorted to ground. This would set the CMP2 output low, but may not be low enough to set the output of CMP1 to high, which turns SW1 off. In this case SW1 would be on, and the device will attempt to transfer power to a shorted bus line. This problem is corrected in the previous circuit, where SW3 is turned off whenever the bus line voltage is below a certain level.

12. Flowchart

Before I conclude this discussion, I will provide a flowchart here that will describe the various concepts introduced in this document about SSB. This is by no means complete and is just preliminary flowchart. Many topics discussed here have been intentionally excluded. For example, I have a complete bit-by-bit description of how data reception takes place, but for transmitting data, there is just one block that indicates “Transmit data”.

The main idea here is just to show how transmission and reception of data takes place and how the addressing scheme for networking works. As each device in the network takes its turn on the bus according to the rotational “taking turns” algorithm, it will go through the sequence described here for transmitting options. The receiving devices will each go through the address decoding mechanism, and if it matches its ID, it will go to the sequence for receiving data.

Each of the devices in a SSB network will use exactly the same software and exactly the same hardware. This is unlike protocols such as USB, where they use different hardware and software combinations for the host, hubs, peripherals, and now the recent “On-The-Go” specification.


START
Set data lines to idle
Keep data lines at idle
BRH?
Set data lines to idle
Transmitter Acknowledge?
BRL?
BRL?
Send address to next in bit-map
Send address to ID + 1
Data Transfer Complete?
Send BRL
Address match?
Increment counter
Reset counter
64 counts?
Assigned to bus time slot?
Data to receive?
Set data lines to idle
Data to transmit?
Reset data acknowledge signal
Receive data bit
Send receiver device address
Transmitter Acknowledge?
BRH
Send BRH acknowledge signal
BRL?
Store data bit
Transmit data
Send data Acknowledge signal
Done?
Set BRH
Acknowledge receiver BRH
Receiver BRH?
Receiver acknowledge?
Set BRH
Y
N

13. Conclusion

This concludes the preliminary discussion of the Simple Serial Bus concept. As you can see, there have been many design phases experimented with in order to find a solution that will work in a real world operating environment. As further development continues, many of the ideas presented here will be modified according to the particular needs at the moment. Hopefully, this document can provide a basic foundation for further development that will create a serial communication protocol that can be easily understood by engineers, technicians, and hobbyists, and to make it relatively easy to implement in their projects as compared to other protocols.

May 28, 2011