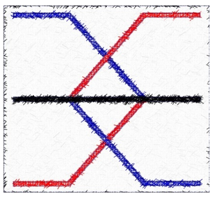


Asynchronous Serial Bus

Concept Description

by Konstantin Dimitrov

Content:

LICENSE INFORMATION

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, known as 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 analogue means are now realised 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 standardised components and possibly from other components that are designed and built from scratch. The 'scratch' components would be anything that is non-standardised 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 standardised components could also be included in this category.

The standardised components would include any part with a specification attached to it, such as integrated circuits, transistors, resistors, etc. This category would also include non-physical objects such as programming languages (C++, Java, Perl, etc.). There are also standardised routines within these programming languages along with several hardware and software combinations that can also be considered as standardised 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, CAN, SPI, I2C, and many others.

As the engineer moves along in the design process, and trying to come up with a system that would satisfy the customer, they wouldn't waste much time in developing a new communication protocol, but would instead use one of the established methods and make adjustments to the 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 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, maximum size of data packets, perhaps the interval for data transfers, the bandwidth needed and many other data fields which only purpose is to describe the device as a 'black box' and let the other party know its capabilities to communicate, where it comes from, etc. Sometimes an engineer would see that many of these advanced features would never be used on his project, in which case they would rather 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.

There are also instances where hobbyists are designing and building a simple digital 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 travelled to a recording device. The 'area of interest' for communications without using USB could easily be extended into any industrial applications, aerospace, various commercial products, etc.

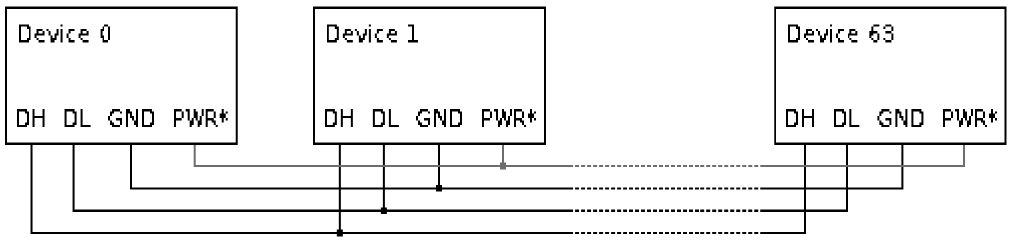
The aim here is to come up with a simple serial interface (Asynchronous Serial Bus) that would replace some of the obsolete methods such as RS-232 and RS-485 and to compete directly with the established USB and CAN interfaces in terms of increased simplicity and reliability, particularly in error detection. 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 their project. The project would usually be designed on a limited budget and probably need something that the designer can understand easily and come to a quick determination as to whether it would perform the needed data transfer in a reliable manner.

Another, usually difficult to satisfy requirement is to accommodate on the same bus many, often very different in speed devices that would need to communicate each with other. In the currently existing serial buses, this is done by sacrificing either speed or simplicity.

ASB is capable of operating in noisy environments, has self-adjusting speed which allows it to perform with flawless transfers even between the slowest and the quickest device, supports up to 64 devices on a shared bus and most importantly is quite simple to understand and implement.

In comparison with USB where the system is required to have a very accurate 48MHz clock and strict timing requirements for the software, ASB does not have any requirements at all which is quite suitable for ultra-low power systems where the communication could go on high speed when possible and on very low speeds, impossible to be achieved by any other serial bus, when conserving power. ASB can even drive directly mechanical switching equipment without the need of additional of buffering the information due to speed differences.

ASB uses three mandatory wires for connecting the devices, plus one optional (marked with a '*' in the picture) for power distribution.



This picture demonstrates the typical topology of ASB. The 'PWR' line is optional (that's why marked with a '*') and every ASB device can safely not use it if not necessary, thus limiting the number of used wires on only three as in the majority of the other commonly used serial buses. By not using the PWR line, however, the device may miss out on the possible backup power supply available from the bus as well as strip other devices from a possible usage of any excess of local power which could be otherwise used to effectively power other devices.

The overall simplicity of ASB would allow an easy implementation in almost any digital system. However there are a few requirements that the bus presents towards the hardware in order to be able to operate properly.

First and foremost, ASB operates by using a bipolar power supply: +5V and -5V must either be available in the system or generated by the ASB hardware. If the optional power pin is used the system must also be able to supply or work from voltages within the full accepted range, specified for 'PWR'.

Another requirement is towards the connector – it must be able to accommodate the maximum configuration of four wires. The connector also must withstand no less than 30V swing with 1A load on the PWR and GND pins.
The most “unusual” requirement to the connector is its connect/disconnect informative ability. If the connector is not mated in a connection, it must keep two of its pins (DH and DL) tied to the GND pin. Upon mating the connector must release mechanically the DH and DL pins from ground before they are physically connected to the bus. This feature is used by the ASB hardware for detecting connection/disconnection events.
It is also required the GND pin to be connected before DH and DL. If the PWR pin is used, it must be connected before the two data lines too.

The following specifications are preliminary and may experience corrections:

	Idle	Active
DH	+2.2V to +5.5V (+5V nom.)	-2.2V to -5.5V (-5V nom.)
DL	-2.2V to -5.5V (-5V nom.)	+2.2V to +5.5V (+5V nom.)
GND	-0.8V to +0.8V (0V nomimal)	
PWR *	+6.5V to +10.5V DC 1A max. (+9V nominal)	

Note: The 'PWR' wire is optional.

2. Data lines and priorities

There are two independent each from other data lines called DH (data “high”) and DL (data “low”). They are bi-directional and serve both as inputs and outputs at the same time. They can be classified as OD-style 'readable' outputs.

Each data line can be at the same time:

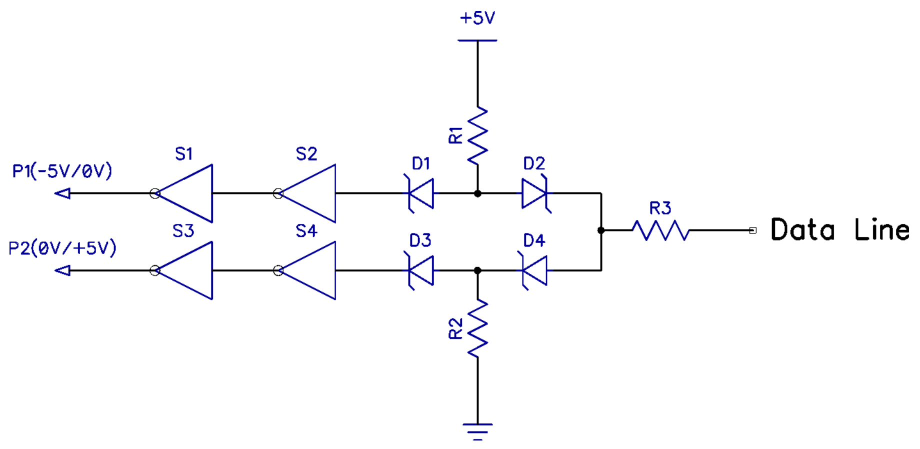
1. Input - capable of recognising three states: -5V, 0V and +5V.
2. Output - capable of being placed into six possible states: -5V (through low or high resistance), 0V (through low or high resistance) and +5V (through low or high resistance).

When serving as an output the actual state of a data line does not only depend on how the device has set it, but it depends on how all devices sharing the same line have set it. There are six priorities defined the data line states:

(highest)	**disconnect**	**SD**	**- DH and DL directly tied to 0V**
	break “high”	**SBH**	**- 0V through low resistance on DH or DL**
	active “high”	**SAH**	**- DH -5V, DL +5V (through low resistance)**
	idle	**SI**	**- DH +5V, DL -5V (thru. medium resistance)**
	active “low”	**SAL**	**- DH -5V, DL +5V (through high resistance)**
(lowest)	**break “low”**	**SBL**	**- 0V through high resistance**

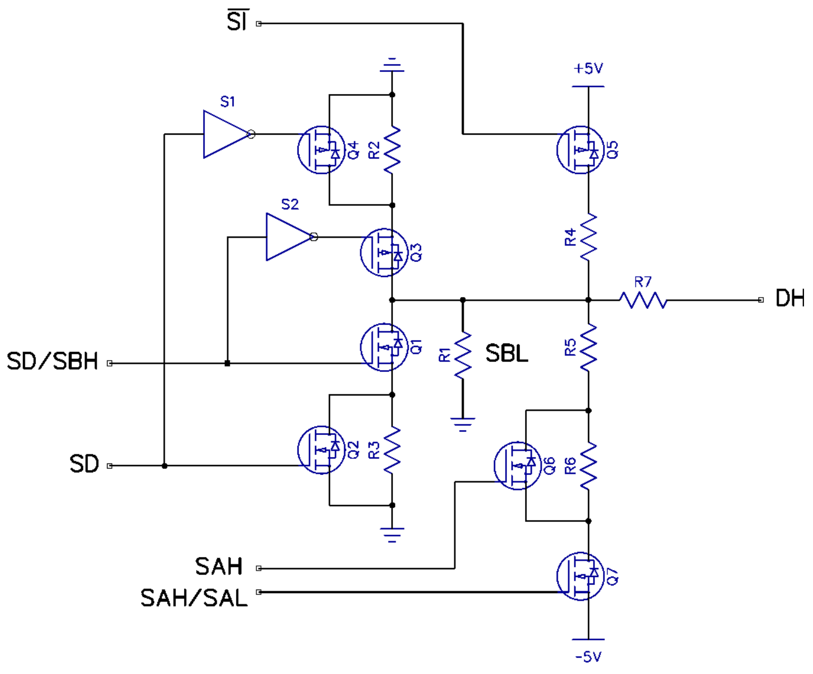
The actual state of a data line is always the highest priority state set by any device sharing the line.

Examples of possible realisation of the circuits needed for a proper ASB communication can be found in the pictures below:



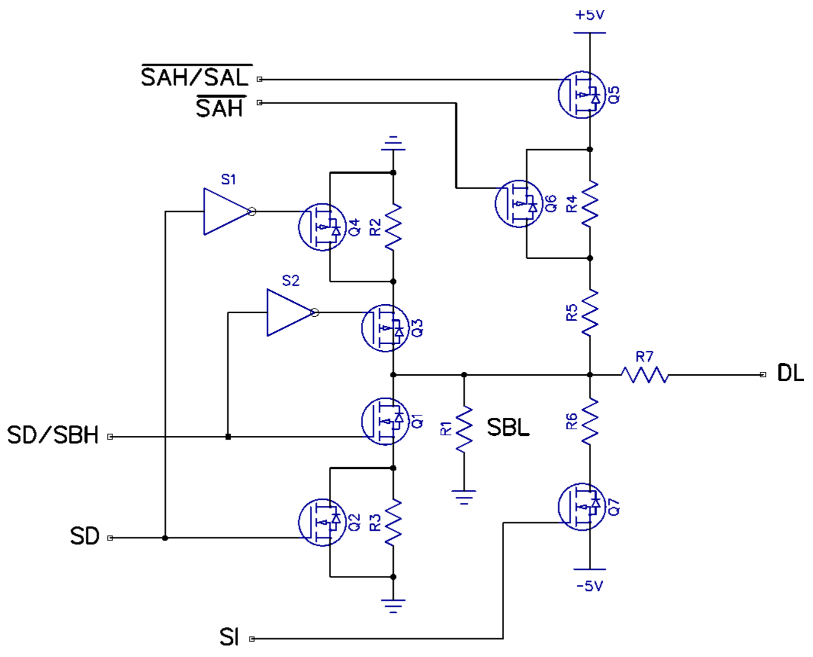
A three-state input (the same circuit is used for both DH and DL lines)

The circuit above would set its outputs in logic states P1='1', P2='0' on 0V on its “Data Line” input, P1=P2='1' on +5V and P1=P2='0' on -5V.



Data transmitter for DH line

For DH line are in place the equations: R1 > (R5+R6) > R4 > R5 > (R2+R3)



Data transmitter for DL line

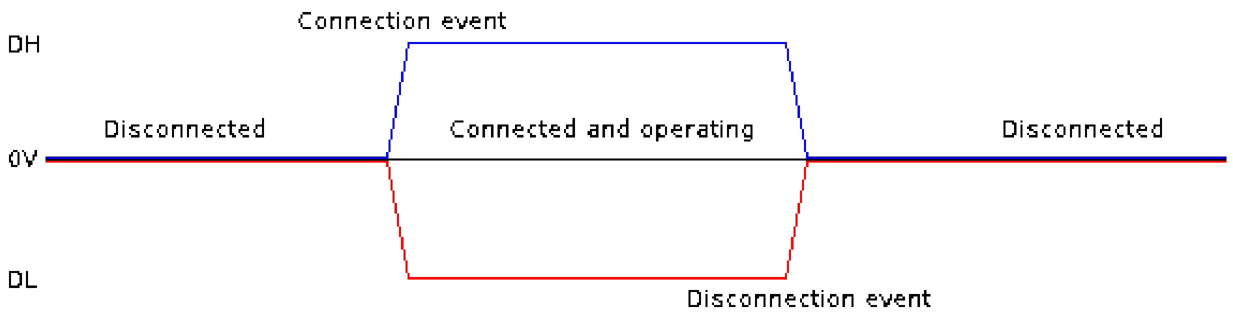
For DL line are in place the equations: R1 > (R4+R5) > R6 > R5 > (R2+R3)

As it can be seen from the pictures the two data lines are basically mirror images of each other and the hardware is quite simple.

3. Connecting to and disconnecting from ASB

Normally when a device is not connected to ASB, its DH and DL lines will be kept at SI by the transceiver and at SD mechanically by the connector. Therefore to the outside world it will appear as in SD level.

Internally the device can determine its current status by reading the DH and DL lines. If they are both set to 0V, the device is not physically connected to a network.



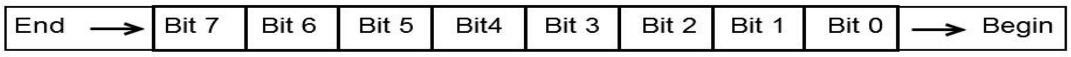
Upon connecting to ASB, the connector will mechanically remove the SD state allowing the device to enter in the network. From this moment on, the device must start responding to all bit transfers but internally ignoring all data until an MBR has been received (see chapters “Break request” and “Networking”).

At any point in time the device could find itself disconnected from the network. Upon disconnecting the connector will mechanically switch to SD state again and the device must cancel any current operation and switch (on transceiver level) its data lines to SI.

4. Basic communication

Once a connection has been established the device is ready to send and receive data. Whilst sending can be performed only during its own turn on the bus (this will be explained later), reception can be performed at any time.

In ASB the information is transferred sequentially at level "bit". If data is grouped into higher order structures (such as byte, word, etc.) the transmission always starts with the lowest bit index and ends with the highest bit index. The picture shows the typical case of transmitting an 8-bit data structure (byte):



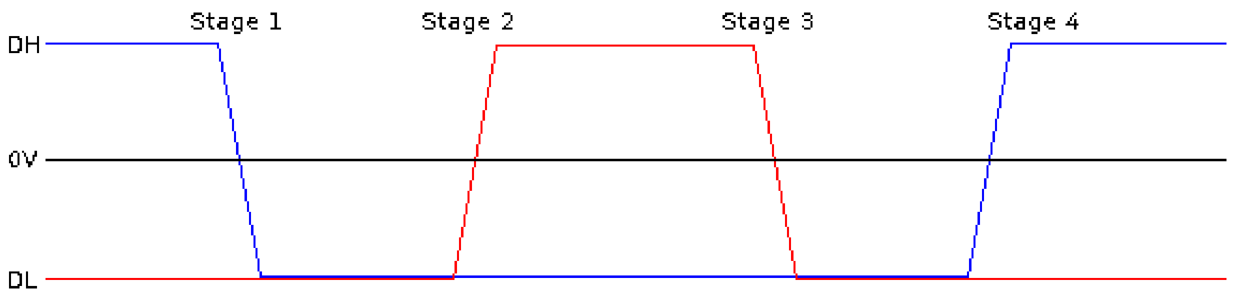
ASB does not require any data padding. The number of bits transferred on the bus equals exactly to the number of bits required. If for instance 11 bits of information are required to be transferred, exactly 11 can be transferred and no padding is made to any higher order data structure. Of course it is completely up to the devices connected to the bus to follow certain data framing, higher order than a single bit.

Any data transfer in ASB can only be initiated by a device called "master" and the master can only transmit (but not receive) information to one recipient called "slave". It will be explained in the following text, but in essence every device takes its turn as master at some point in time and acts as a slave at any time when is not master.

At hardware level the two data lines have distinctive roles. The DH line is always used to transmit bits with value '1' and the DL line is always used to transmit bits with value '0'. During transmission the opposite of whichever data line has been used by the master, becomes a handshaking line and is used by the slave to confirm the information.

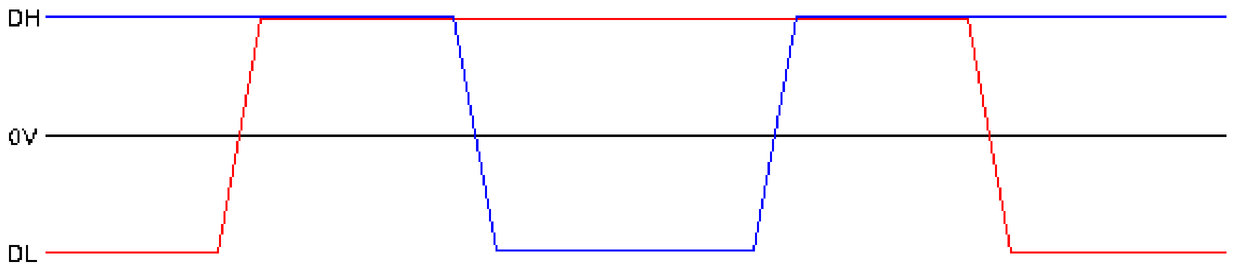
Any transmission starts assuming the data lines both for the master and the slave are in SI state and they must return back to SI after the transfer has ended.

Down to a single bit level, transmission is made in four distinctive stages:



This picture shows transfer of a bit '1'.

The same rules are in force for bits '0' but the whole picture is simply reverted, starting with DL going active in stage 1 and the relevant mirror image in all the following stages.



Stage 1:
The master initiates the transfer by setting either its DH (in case of transmitting a bit '1') or DL (in case of a bit '0') in SAH. The line used for transmitting the actual information will then become the "active" line. The other line automatically takes the role of "handshaking" line.
Along with setting up its active line, the master also must place its handshaking line in SBL.

Since both DH and DL are initially kept in SI by all devices, by setting the active line in SAH, the master effectively changes the line level to SAH while the handshaking line will remain in SI at this moment because SBL has lower priority than SI.

Stage 2:
Upon detecting one of the lines changed from SI to SAH, each slave device knows there has been a data transfer initiated on the bus which indicates a bit '1' in case of DH being in SAH or '0' in case DL being in SAH.

The slave must place its data lines from SI to SAH for the active line, and from SI to SAL or SBL (this will be further explained in chapter "Networking") for the handshaking line.
Since both SAL and SBL states have lower priorities than SI, the handshaking line will effectively be kept SI until all slaves have confirmed the transfer with SAL or SBL (the master's line is in SBL at this time and is not affecting the handshaking line state).

Stage 3:
After initiating the transfer in stage 1, the master must only keep checking the status of its handshaking line. When changed from SI to SAL or SBL, the master knows all slaves have reacted and confirmed the transmission.
At this moment the master responds by switching two data lines back in SI. This will only change the state of the handshaking line but the active line will remain in SAH, held by one or more slaves.

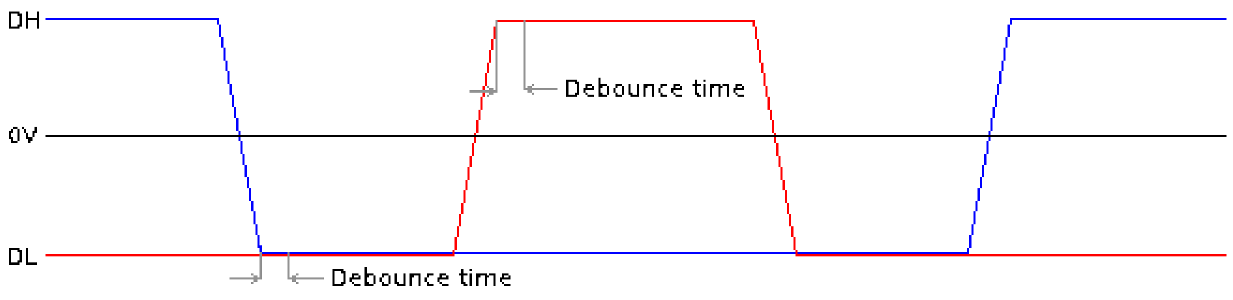
Stage 4:
After responding to the initial transmission in stage 2, all slaves must keep checking their handshaking line for a change from SAL/SBL to SI. Then they must release their two data

lines from SAH back to SI. Since SAH has higher priority than SI, the active line will take SI level only after all slaves have set it in SI.

Aftermath:
At this moment of time both DH and DL are in SI, the slave(s) have received a bit of information from the master and the master knows that. Another transmission can take place immediately.

ASB does not impose any minimum reaction times for the devices. A data line can be held in any state indefinitely. This makes ASB very suitable for connection devices which speed can vary in limitless range. Another effect of that is the transfer speed is self-adjusting at any bit.

ASB uses level-sensitive logic. Upon any transition of a data line, a certain programmable minimum time must elapse in order to let the logic accept the current state as valid. This period is called "debounce" time and is used to prevent the logic from accepting as valid any accidental changes in the level of a data line in result of any digital noise. Of course if the debounce time is set to 0 the device will effectively be using edge-sensitive logic.

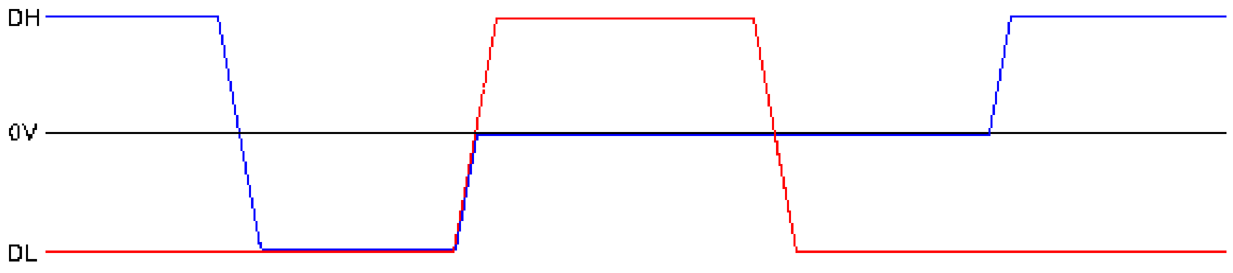


5. Break requests

During transfer at any time either the master or the slave may experience the need to inform the other party about its inability to take more data. In this case a special even on the bus is occurs and it is called a "break request".

It can originate from one or more slaves and is called "SBR" (slave break request).

The picture below shows the process of SBR during a transfer of bit '1':

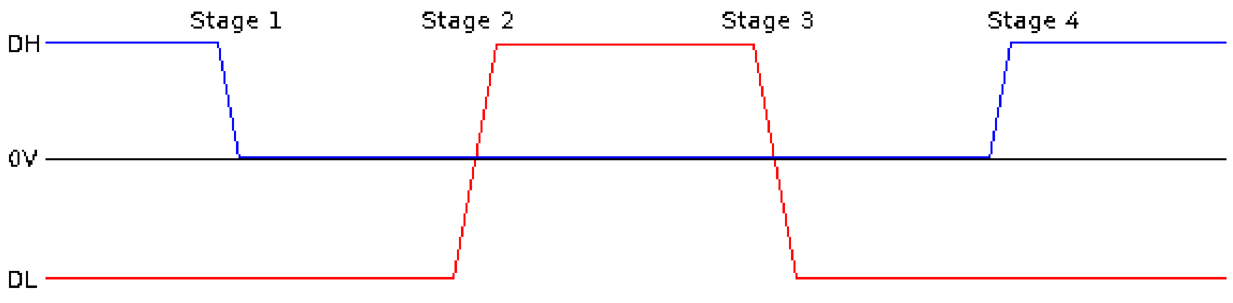


It is similar to the normal transmission in most of its time.
The master initiates the transmission by setting the DH line is SAH and the DL line in SBL. Now any slave device on the bus is supposed to confirm the transfer normally by setting their DH in SAH and DL in SAL or SBL.
But if a slave needs to generate SBR it must set its active line (DH in this example) in SBH, instead of SAH.
SBH has higher priority than SAH and will effectively change the line level. Slave sets its handshaking line as in a normal transfer in SAL/SBL.

At this stage the master would detect the change in condition for the active line and recognise it as SBR. The rest of the transfer follows the normal pattern – master releases its two lines back in SI which would only change the state of the handshaking line.

Then all slaves must release their lines in SI, which will remove the SBH from the active line and return the bus back to initial state.
Now since the master has received a SBR it will immediately terminate any further transmission.

A break can also originate from the master. In this case it is called "MBR" (master break request).



The picture above demonstrates how the MBR process is going.

The master wants to inform all slave devices about the termination of its transfer.
Instead of initiating a new transfer, it will place one (either DH or DL are acceptable as active for an MBR, in this case the picture shows DH) of its data lines in SBH and the other one in SBL.

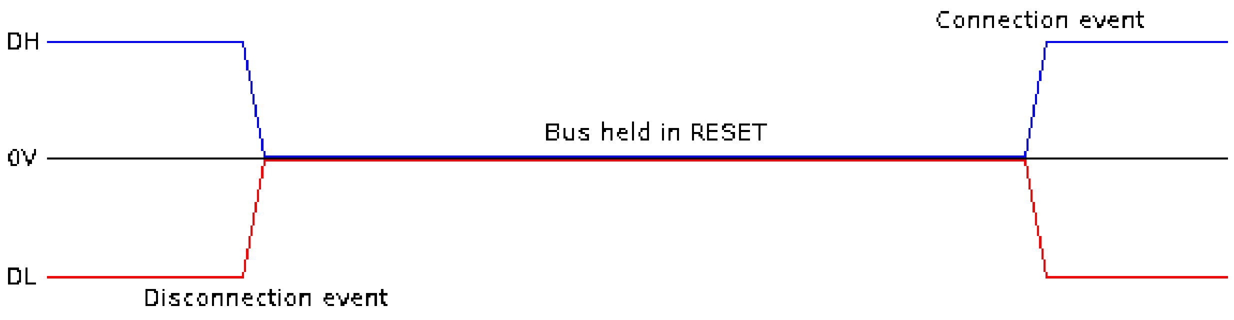
Slaves will detect the change on the active line and confirm it by placing their active line in SBH and the handshaking line in SAL/SBL.
When all slaves have done that the master would detect the handshaking line (DL in this example) changing from SI to SAL (or SBL).

Now master releases its two data lines back in SI. Slaves detect the change in the handshaking line from SAL/SBL back to SI and release their two data lines back in SI.

When all slave devices have done that the bus returns in its initial state.

Since ASB does not have any strictly defined protocol about the data going through the bus, SBR and MBR are very important and the only way to allow the communicating devices inform the others about end of a transmission.

In emergency situations the network may need to be reset back to its initial state. A special event called "bus reset" can be used to to that. Reset can be generated by any device at any time. It basically simulates all devices being disconnected and the re-connected to the bus. The timing diagram below shows how the reset event can be generated by placing both the data lines in SBH at the same time, hold them for some time and then release back in SI.



6. Networking

ASB allows more than just two devices to share the same data lines. In order to do that every device must carry some unique identification sign to let the others address it when necessary.

Every ASB device has a 6-bit static “id” either pre-set or programmable by the user. Within the scope of a single ASB network, every device must have its id set to unique number.

Device with id 0 is always required to be present on the network.

In addition to the id, every device has its own “id map”. This is a 64-bit internal register which uses one bit of each position to indicate the presence of the device with that id number on the network – bit 5 refers to device with id 5, bit 21 to device with id 21, etc. If a bit in the id map has value '1' that indicates the relevant device is present on the network, otherwise if the value is '0' the device is missing and cannot be addressed.
An id map is always reset with all values '1' upon a device connecting to the network.

All devices take turns to be master on the network. These turns are called “mandate”.
During its mandate a master can only initiate outgoing transfers (only send) to one or more slaves. Then the master is required to nominate a next master to take over the bus.

Every transfer initiated by the master follows the same format – a 6-bit id of the receiver, possibly followed by data and MBR.

Let's take a look in more details.

First of all, any device acting as master should be able to reliably address any other device, acting as a slave.

After a power on event or connecting to the bus, a device doesn't have any information who is online and who is not. All bits in the id map are '1'. The device would only listen and ignore any information until a MBR event. At this moment in time the communication is synchronised and the device can start receiving information or sending it when it mandate has come.

The first 6 bits following every MBR are considered as device id.

All slave devices start receiving the id header bits and compare each bit individually with their own id. In case of a match they would set their handshaking line in SAL.
If the current bit does not match with the relevant bit the their own id, the device must set its handshaking line in SBL. From this moment on this device would only listen and confirm transmissions but ignore everything until a new MBR.

This is an example of the addressing process:

Device 0 (currently master) is trying to address device 22 (0x16). There are devices with is 0, 11, 14 and 22 present on the bus.

Id 0x16 is represented by the bit sequence '010110'. The transmission always starts with the lowest bit first, so the master would send a '0'.

Device 11 has digital representation of its id in the form '001011'. Currently checking bit in position 0 which in its id has a value '1' but the master requires '0'. So device 11 fails on this stage and responds with SBL.
Device 14 has digital representation of its id in the form '001110'. Its bit 0 has value '0' so device 14 is still in the game and responds with SAL.
Device 22 has digital representation of its id in the form '010110'. Its lowest bit is also '0' and device 22 responds with SAL.

After this stage the master finds SAL on the handshaking line and continues further with the next bit from the id which is '1'.

Device 11 has already failed and only confirms with an SBL.
Device 14 confirms with SAL because its second lowest bit is also '1'.
Device 22 also confirms with SAL due to the match of its bit with what the master needs.

Master finds SAL again on the handshaking line and continues with the bit on position 2. The bit is '1'.

Device 11 has already failed and only confirms with an SBL.
Device 14 again confirms with SAL because its bit is also '1'.
Device 22 also confirms with SAL.

Now the master requires a bit '0' as in bit 3 from the addressed id.

Device 11 has already failed and only confirms with an SBL.
Device 14 fails here and returns SBL.
Device 22 confirms with SAL.

Going further the master will send all the six bits and device 22 will confirm. So after the sixth bit the master will still be receiving SAL which is an indication that there is a device with the requested id and that device is ready to communicate.

If there wasn't a device with the requested id, the master would have found SBL after addressing some of the bits because all slave devices would have failed by then on on that one.

From master's point of view, after addressing a device the handshaking line would change from SI to SAL if there is at least one slave device that has confirmed the current addressing bit as valid. Otherwise the master will find the handshaking line in SBL which is an indication that there is no slave device that has confirmed the current bit as valid (or no slave devices at all). The master must terminate the current transfer by issuing MBR, clear the relevant bit (according to the id, it was trying to address) in its own id map to mark the lack of device at this address and continue with either a new transfer or nominate a new master.

Slaves only use SBL during the addressing phase to indicate a mismatch in the current addressing bit with their own one. In any following transfer only SAL should be used as SBL doesn't have any meaningful role outside of the addressing.

If there has been a slave device that has confirmed the address, the master continues with transfer of data and issues MBR at the end of the transmission.

The master tries to address only those devices which are marked with '1' in its own id map. The id map is reset back to all bits '1' on every 1024th mandate of a device as master.

Resetting the id map would allow any recently connected devices to take over as masters or being addressed as slaves.

During a power on event always the device with id 0 takes the role as the very first master on the bus. The first thing it is required to do is issuing a MBR. Thus it will synchronise all the currently connected devices and let the network start with its first mandate.

After finishing with all data transfers the master is required to nominate the new master. This is done in similar fashion as a normal data transfer with the only difference there must be MBR immediately following the id after the slave device has confirmed its availability. If the slave device detects MBR before any data coming from the master, it takes the role of master and the old master becomes a normal slave device.

The current master always nominates next masters only amongst the devices marked with '1' in its own id map. The device with the closest bigger id that has confirmed its availability gets the nomination. The address id's roll over 63 back to 0.

If any device has failed to confirm its availability during the addressing, it immediately receives a '0' bit in the id map of the master.

Simplifying the text above, there are two forms of communication in ASB:

1. Master send information to a slave:

Slave id	At least one bit of information	MBR

2. Master nominates a new master:

New master id	MBR

7. Power distribution

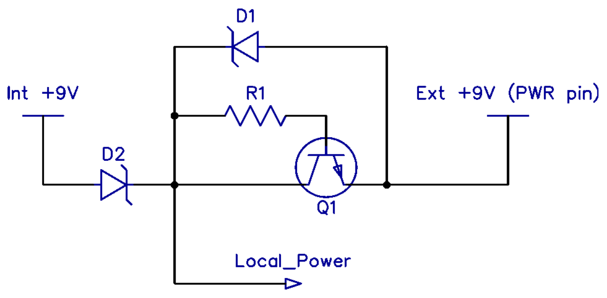
Occasionally a device which is connected to the bus may find itself without internal power. In such cases ASB provides the option to let the device draw power from the bus via an optional fourth wire.

The presence of power on the 'PWR' line is not guaranteed.

For power distribution ASB follows the rule “If I have more than I need, I could share. If I need some, I hope someone would share with me.”

This in essence means that every device that is connected to the power wire could provide power to other devices but also could draw power if it needs to. But providing power is only as long as the power supply of the device allows it without any significant voltage drop.
The principle is similar to several linear voltage converters connected in parallel. The voltage on the common bus would roughly be equal to the highest voltage generated by any one of them. Thus the highest voltage potential power supply would find itself providing power over the bus. If there are a few power supplies with very close potentials, they will unequally share the current depending on their potentials.

There are many ways to achieve this goal. Probably the easiest one would be a uni-directional current limiting clamp as in the picture below.



This solution allows the local power to be produced by either an internal power supply or coming from the 'PWR' pin on the connector. If there is an external source present the current would flow through D1. The device could also source power through the 'PWR' wire to an external sinking device. In this case the current would be limited for the outside world by the pair R1/Q1 which effectively set the maximum current the device would be able to share.

This circuit is a simple example only and it does not take in count the voltage drops occurring in the components. In the real life the voltage sourced onto the 'PWR' line will be lower than the bus voltage but as far as it falls within the specified limits, it can be considered as valid.

Of course much more sophisticated solutions could be used to minimise possible excessive draw of current by a faulty device.

8. Basic extensions to ASB

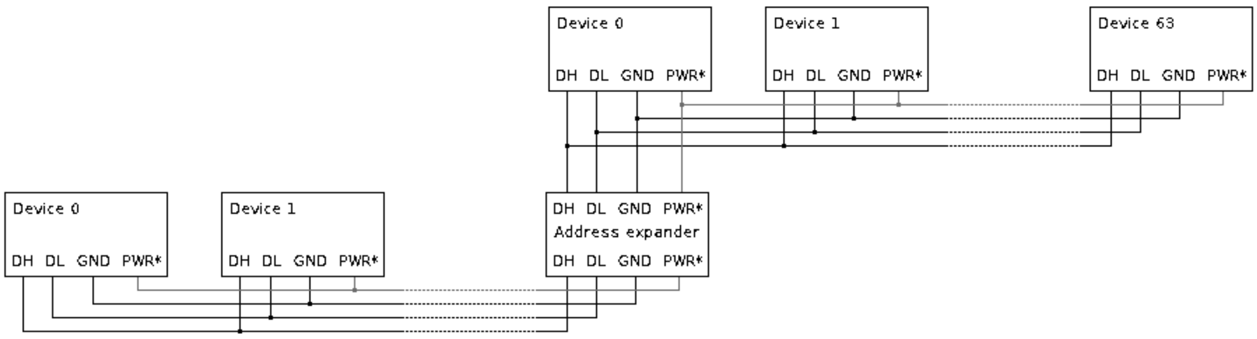
8.1 Watchdog

Even being completely asynchronous and open towards any possible rate, ASB is not completely immune against any possible fault that might happen in the real life. As examples that could be a data line, indefinitely held in some state or the current master device failing or being disconnected during its mandate.
In general the whole communication relies on the proper work of the current master and if it fails the network may find itself built from slaves only and there won't be a master to initiate the next transfer or nominate a new master.

A simple extension to ASB could resolve the situations above with very little trade-offs.
One or more special devices called "watchdog", connected on the bus with the only function to wait for being nominated as master. When that happen the watchdog only nominates the next master and resets its internal counter so it knows the bus is functioning normally.
If the watchdog did not get the role as a master during some time it could assume the network has stalled and generate a bus reset event thus allowing the communication to start over as in power on event.

8.2 Address expander

Although ASB is capable of directly accommodating up to 64 devices on the same bus, is some rare occasions that may not be enough. Another special function device would allow devices to communicate and address each other beyond the 6-bit id barrier. Of course this will require further extensions in the communication protocols as well.
Basically the concept is to have a device that connects two networks together as in the picture below.



The address expander looks as a single device in both the networks but acts as a bridge between them. The software protocols must use extended 12-bit id headers when addressing the expander in order to access devices from the other network.

9. Conclusion

ASB could emerge as a viable alternative to many of the today's serial buses, including CAN and USB. It's simplicity allows implementation even into the lowest end microcontrollers without increasing the cost of hardware or taking space in their memory for any software stacks.

With the development proceeding further there might be changes in some details described in this document but the main principle of work for ASB would remain the same and big winner at the end will be the user, benefiting from the lower development and hardware cost of the devices using ASB.