Wireless Binding Methodologies

Author: Brian Booker

Associated Part Family: WirelessUSB™, PRoC™

Associated Code Examples: None

Related Application Notes: None

To get the latest version of this application note, or the associated project file, visit http://www.cypress.com/go/AN6066.

AN6066 discusses the general considerations for deciding on a binding methodology of devices in robust wireless systems, and also gives examples of common binding schemes, highlighting their advantages and disadvantages. It is intended for system architects, particularly those who are relatively new to wireless designs.

Contents

1 Introduction

Robust wireless systems require a well thought out method of establishing a connection between different elements in the system. This method is called “binding”; its purpose is to enable communication and prohibit unwanted connection with devices outside the intended system.

When connecting a wired peripheral to a host computer, ‘binding’ is a straightforward process. Essentially, if the cable can be detected, then the peripheral is recognized and therefore, ‘bound’ to the host. For example, in a USB peripheral, when the USB cable is plugged into the host PC, the cable connection is automatically detected through the D– or D+ pull-up resistor. The host sets a device address during enumeration, which is used to communicate with that specific peripheral in all further exchanges.

In the case of wireless systems, binding is a more complex process. Since there is no physical connection between elements, other methods must be employed to indicate that a connection is implied or desired. This is what we refer to as ‘binding’. This application note discusses general considerations for establishing binding schemes and describes several methods for binding in WirelessUSB™ systems. Specific code is not provided—for firmware, it is best to go straight to the Cypress wireless reference designs.

In short, binding determines which RF elements should be able to communicate with one another. This is essential in environments where multiple wireless systems may exist in close proximity sharing the same RF band to communicate. A host may be able to hear wireless communications from many peripherals within its proximity, but it needs to be explicitly told which ones to listen to and which to ignore.

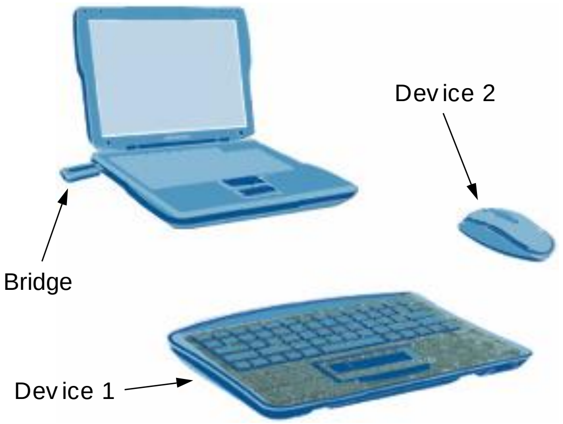
2 General Considerations

This section presents the definition of terms and the system requirements for wireless binding.

2.1 Definition of Terms

Wireless systems are very flexible in their topology, with a specific solution typically being defined by the firmware implementation. Many of the systems that Cypress supports are PC-centric with one or more devices having an access point to a single PC. We will generally consider this topology, but the concepts discussed are also applicable to other topologies.

Figure 1. Typical Wireless System



Bridge is the term that is used to describe the access point. In a PC-centric network this is usually a small form factor device that attaches to the USB port. Thus, it ‘bridges’ the data from the remote device to the PC. In embedded systems it may attach to another processor via a UART, SPI, or similar interface—or it may be the embedded device itself. In either case we will consider the center of the system.

Device is the term used to describe the element that is being added to the network. Examples include a mouse, keyboard, remote control unit, temperature sensor, actuator, and others. Typically, there will be one bridge, but one or more devices in a system.

The term binding has already been introduced, but be aware that other terms are also used in the industry to refer to essentially the same process. ‘Pairing’ is one of the more common terms. Cypress uses the ‘bind’ term to imply that the connection is generally of a more permanent nature. Usually, devices are bound once and there is no need to repeat the process during the life of the product. This may not be true in all applications.

2.2 System Requirements

This application note introduces some common binding methodologies, but in reality there are many variations on these themes. Choosing the right binding method is all about satisfying a particular end-user experience. There may be some give-and-take in this process, but defining the desired system and user goals will help to converge on a suitable implementation.

The Cypress solutions have multiple strengths. The first is that Cypress uses a two-way communication between different points in the system; thus, every transmission is acknowledged by the recipient. Cypress also uses a robust network protocol producing the industry's best interference immunity, as well as support for a vast number of co-located systems. This is accomplished in part by using diversity of channel, pseudo-noise (PN) code, and checksum seed to establish communication between elements on a given network. Some applications may choose to use other parameters—the Device ID assigned by our CY3635 N:1 kit is a good example. These are the parameters that must be established at bind time.

In order for the two elements to communicate for the first time, and establish the intent to join or form a network, they must be able to find each other using a common channel, PN code, and checksum seed at a specific point in time. This is the complicated problem that this application note endeavors to explore.

It is worth mentioning that robust protocols and binding schemes may not be necessary for all applications. In some cases, it may be perfectly acceptable to fix one or more of these values, such as channel, or even to allow limited selection such as through a switch or pin strapping. Many of the console game controllers use a similar method today. Even one-way communication may be suitable in some systems. In general, these might be appropriate for non-consumer applications where the environment is well controlled and understood. These are simplistic enough that they will not be explored further in this application note.

2.2.1 User Action

For most applications, a few factors should be considered. The first is user action. To start using this new device or to add it to an existing system, what action must the user take? Press a button? Follow a power-on sequence? Launch a software application? Try to use the device in a ‘normal’ fashion? Of course, the goal is to make the process as simple and intuitive as possible—both to make it easy for the end user and also to reduce the cost of customer support calls when things do not go as expected. But ‘intuitive’ is a subjective thing, and may vary from application to application.

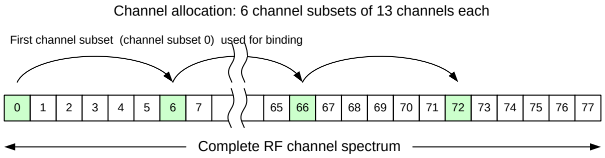
2.2.2 Sequence

The next thing to consider is whether or not a specific sequence should be followed. Limiting the process to a specific user sequence of events (for example, first press the bind button on the bridge and then press the bind button on the device) can potentially simplify the process for the developer, or allow more flexibility on other aspects of the solution. It may also help to reduce or eliminate the timeout that is typically required—at least on one side of the system. This means that inadvertent activation of the bind process on this side may be almost imperceptible to the end user. The disadvantage is that the user has to remember or look up what the right sequence is. It also increases the likelihood of a technical support call to resolve the situation.

2.2.3 Protocol

Another consideration is related to protocol and network parameters. Again, the assumption is that customers will take advantage of the Cypress solution advantages, but that does not mean that our exact code must be strictly followed. The primary goal of binding is to establish the right set of network parameters to enable ongoing communication. But how do you know where to initially go to conduct the bind sequence? The simplest method is to use a default set of parameters: for example, channel 0, default PN code, and 0 checksum seed. But what if there happens to be strong interference on channel 0 at the time? It may not be possible to successfully conduct the binding operation. Other alternatives are to introduce some level of protocol to enable variation of certain network parameters (typically the channel) to provide increased opportunities for the elements involved in the bind process to establish that initial communication.

Figure 2. Typical Implementation of Channel Subset for Binding



In these examples, we also assume that the device is the one actively transmitting that is attempting to get permission from the bridge to join the network. The bridge is the default receiver. Remember that there may be applications where a different scheme may be more appropriate. For example, a bridge may periodically broadcast its network parameters, say once every few seconds, so that any device that is brought into range of the network can identify it and present the user with an opportunity to join.

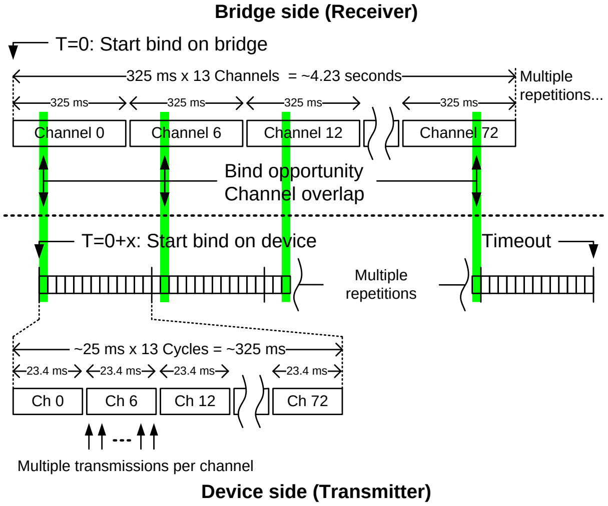
2.2.4 Timing

There are three aspects of timing to discuss. First, how long does the user have to take the necessary actions? The bind process typically interrupts normal device operation. A timeout (as brief as possible) is desired to handle cases where the bind fails or when it was inadvertently activated, but it needs to be long enough to be usable. Consider a system targeted at desktop PC users. The USB port may be on the back of the PC under the desk. Enough time should be allowed to crawl under the desk, press the button on the PC, and then crawl back out and press the button on the device side—maybe 20 seconds.

The next thing to consider is how long until the user sees a response from the process. In many cases, the response may be nearly instantaneous, but more complex binding schemes that allow more flexibility or more handling for interference may require additional time. A few seconds without a visual response may be enough for the user to be confused and try to start the process again, even though the first attempt may still be in process.

Finally, there are low-level timing considerations. In particular, this applies to more robust schemes where network parameters are being varied to try to give a better chance to bind in the presence of interference. If you allow changing of channels (or other parameters), you must carefully think through the sequence to make sure that there will be adequate time for both elements to find each other in a common place. This affects two things: the probability of successful completion of the process and the length of time required for the user to see a response as described previously.

Figure 3. Bind Timing Example



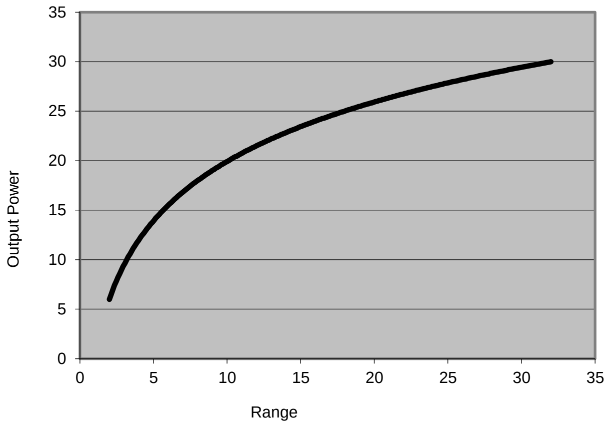
2.2.5 Range Discrimination

Many products may support multiple co-located networks that are potentially in communication range of one another. Examples are keyboard/mouse products in an office environment, or building sensor networks. During the bind process there may be a risk of cross-binding—accidentally binding the device into the wrong system/network. It may not be obvious to the user what has happened or how to fix it. Limiting range can help to constrain this problem. One of the other likely places to encounter this issue is when trying to bind devices on the production line. With multiple units being bound in rapid succession and close proximity, there is an increased potential for cross-binding.

Limiting range is usually not a problem for the end user. For most systems, it is reasonable to assume that when configuring the system the user has the device to be bound in close proximity to the bridge. Limiting range can provide more control over the binding process. But there are applications where that might not be the case, such as with sensor networks in a building control system. In this case, maximum range may actually be helpful to the bind process.

The tricky part of this is that, although the user perceives distance, the wireless device generally does not have the ability to measure this directly. The wireless device senses signal strength, which can roughly correlate to range. Many factors can impact this, such as actual transmit power, interference, attenuation through walls or other material including product enclosures, multipath effects, antenna orientation, and others. So although it may be desirable to limit range generally this can only be done at a coarse level. As a simplification (considering only free-space signal spreading) a 6 dB delta in power roughly correlates to a 2X delta in range.

Figure 4. Typical Relationship of Output Power vs. Range



2.2.6 Calculation of Network Parameters

An example of key parameters that come from the binding process was provided earlier: channel, PN code, and checksum seed. The actual requirements for the parameters may vary by application—how many nodes are in a particular system, how many systems could potentially be co-located within communication range, and so on. Cypress has a couple of examples, the best of which comes from our two-way HID protocol. More details can be found in the CY4636 User's Guide.

What source is used to generate these parameters? For most applications, a random distribution across the target parameters is desired. Cypress typically uses the Manufacturing ID (MID) contained within the radio as a seed for calculating these numbers. This is a 6-byte value and, although not guaranteed to be unique, is unique enough for most applications to use it as a source for creating a diverse array of wireless networks

In particular, the MID from the bridge device is typically used, because it is the central point in the network. A common algorithm is applied to all devices in the system; so if multiple devices are provided with the same MID, which is exchanged during the binding process, then all devices will arrive at the same selection of network parameters.

Figure 5. Sample Algorithm for Network Parameters

6 byte Manufacturing ID

MID 6	MID 5	MID 4	MID 3	MID 2	MID 1

PN Code = (MID1 << 2) + MID2 + MID3
Base Channel = (MID2 >> 2) - (MID1 << 5) + MID3
PIN = (((MID1 - MID2) & PIN_MASK) + MIN_PIN)
CRC Seed = ((MID2 >> 6)) + MID1 + MID3

Refer to the CY4636 User's Guide for a more thorough description of all parameters

In some cases, it may be desirable to use a source other than the MID to seed the calculation of network parameters. Allowing a value to be specified, such as through a Windows dialog box, may allow a higher degree of end user control on the setup of the network. In other systems there might even be multiple bridges, such as with media center remote controls. A household may have one or more remote controls that interact with one or more PCs and set top boxes. In this case, the system needs to account for the fact that there are multiple bridges, so the firmware may need to base the network parameters on a default bridge MID and allow overriding of the MID for the second, third, fourth, and other bridges in the household. Or it could establish a completely separate mechanism for establishing the seed value for the algorithm.

2.2.7 Storing Binding Parameters

There are two somewhat related topics here: where are the network parameters stored, and is there any way to get rid of them and start again?

In general, the network parameters or the seed for the algorithm will be stored in nonvolatile memory. That way users do not have to rebind their systems any time power is removed—for example, to replace batteries or when the bridge is removed from the USB port of the PC. For most of Cypress's examples, the information is stored by the device and not by the bridge. This allows many devices to communicate with one bridge without the bridge having to keep track of all devices. The bridge will talk to any device that knows its network parameters. The bridge's MID is used as the seed for the network parameter algorithm, so each device must store the MID of the bridge that they want to communicate with. The actual network parameters can be recalculated as needed from this base. The implication of this is that the bridge does not know which devices will talk to it unless additional firmware manages this. For some applications it may be desirable to have the bind parameters stored on the bridge side, or even on both sides.

The other question is "Can I un-bind?" For some systems this is unnecessary. A system using a button, for example, can initiate an attempt to bind with a new bridge whenever the button is pressed. But some systems may not use such an explicit mechanism. For certain applications, it may be useful to provide some other mechanism for removing the previous bind parameters: removing and restoring power, a Windows application, or a special button.

2.2.8 Controlling the Number of Bound Devices

One last thing to consider is whether there needs to be a control on the number or types of devices that are bound. The Cypress HID design provides a good example. Parameters are stored only on the device side and not the bridge side. Therefore, it is possible to have a large number of mice and keyboards all talking to the same bridge. In some cases this might be good. For example, consider adding a presenter tool to the product portfolio. These tools typically appear as keyboard-type devices (page up, page down, esc keys, and so on). The presenter can easily be added to the system without any changes to existing product firmware, enabling it to appear similar to another keyboard. But this flexibility also might make it easier to inadvertently cross-bind an unintended keyboard into the system.

3 Binding Methods

Having discussed some of the key concerns in determining a binding mechanism for a particular system, we can now examine specific implementations. The discussion will assume that the capabilities of WirelessUSB LP are being used, but the same techniques can also be applied to WirelessUSB LS, LR, or other wireless systems, although the packet structure and content may vary. Here is a typical example.

The bridge device (the one controlling the binding) must be in a receive mode listening for a request. The device that is to be bound into the system sends out a packet indicating a bind request. Typically this is indicated by a particular byte, such as the packet header. Other information can be passed in the remainder of the packet if required by the application. When the bridge receives the request, it acknowledges it and sends a subsequent packet with the necessary data used to calculate the network parameters, which is then acknowledged by the device. At this point, both sides of the system have the information to establish communication with the same set of network parameters so normal operation can begin.

3.1 Factory Binding

Although often overlooked, factory binding can have a big impact on the end user.

3.1.1 Description

The vendor supplying the product prebinds the devices during the manufacturing process and ships them together in the same package. Therefore, on receiving the product, the user can begin using it without any other action required. Factory binding usually does not exist on its own. Another binding method is typically provided with the product to enable the factory bind process and also as a backup for the end user. It is also possible to have outgoing test software that preloads the necessary values into nonvolatile memory.

3.1.2 Advantages

- Easiest method with highest chance of successful user experience. True plug-and-play. It just works out of the box
- Least risk of technical support calls, therefore smallest support cost
- No impact to power consumption in battery-powered systems

3.1.3 Disadvantages

- Risk of cross-binding on the manufacturing line
- May require additional manufacturing steps, which adds a small amount of time and cost
- Not suitable for systems where additional devices can be sold independent of the bridge

3.2 Two-Way Button Bind

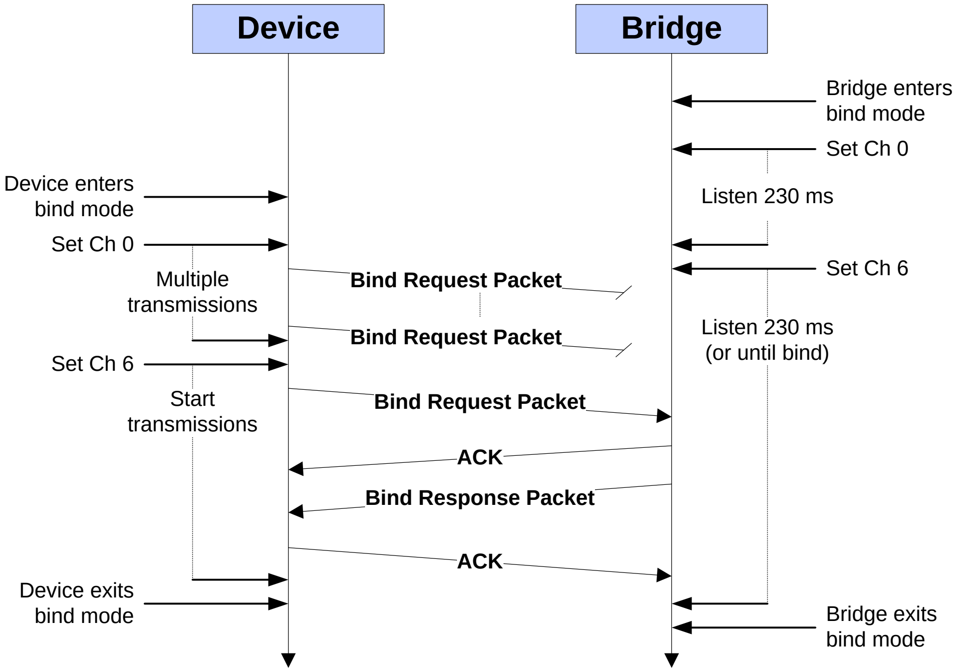
This is a common method of binding, which is used on most Cypress WirelessUSB systems. It is straight forward and easy to use.

3.2.1 Description

Each side of the system (device and bridge) uses a specific button to initiate bind. In this scenario, there is no required order in which to press the button. When the user presses the bind button on the bridge, it is placed into a listening mode, ready to receive bind requests. When the button is pressed on the device side it begins continuous transmission of bind requests. Both sides have a timeout of about 20 seconds, allowing ample time to press both buttons in case one is not in an easily accessible location. Furthermore, multiple channels are used to increase the chance of a successful bind. Both the bridge and the device must rotate through the defined subset of channels. A reduced output Power Amplifier (PA) level is used, which slightly limits the range but does not tightly constrain it. When the bridge receives the request, it responds with the MID used to seed the network parameters algorithm at which point the sequence is complete.

Timing is similar to that shown in Figure 3. Sequencing is shown in the following figure.

Figure 6. Example Button Bind Sequence



3.2.2 Advantages

- Simple system resulting in good user experience. It is reasonably well understood by most users because it is commonly used
- Minimal risk of technical support calls
- Adequate timeout ensures users can get both bind buttons pressed even when one device is not quickly accessible. Buttons can be pressed in any order

3.2.3 Disadvantages

- Small cost impact for button hardware on both sides
- Ties up the system when the bind button is pressed due to the timeout required. This can be problematic in the case of inadvertent bind button presses
- Inexperienced wireless users may not understand the bind requirements and the need to press buttons before using the device for the first time—small potential for support calls

3.2.4 Derivatives

Software-initiated Bind on Bridge

Most of the systems discussed connect through a central bridge to another processor (PC, set top box, and others). Instead of using a physical button, it is possible to command the bridge into the bind mode through a software interface on the host. This just implements a ‘soft’ button so the process is otherwise exactly as described above.

Additional advantages

- Cost savings through elimination of button

Additional disadvantages

- Not obvious—requires user documentation and increases risk of support calls
- Increased development time/cost for host software

Reduced Timeout—Sequencing Requirement

There are various uses for a specific sequence, but the main one is allowing one side of the system to have a drastically reduced bind timeout. The other side typically retains the long timeout to allow the user adequate time to initiate bind on both sides of the system. Bind is initiated first on the side with the long timeout and then on the other side. Since the second side knows that the first side is already in bind mode, it can expeditiously execute its own sequence, probably fast enough that any delay is not noticeable by the user. Completion of binding will appear almost instantaneously after the second button press. A good example is a very small form factor bridge where the bind button may be accidentally pressed when inserting the bridge into the PC’s USB port.

Additional advantages

- Reduced timeout on one side—inadvertent bind button presses are imperceptible by user
- Bind power consumption on that side can be reduced

Additional disadvantages

- User must remember the sequence or bind will fall

3.3 Power-up Bind

At the highest level, power-up bind means using the power-up process as a replacement for pressing a button.

3.3.1 Description

Every time power is applied to the unit with power-on bind implemented, it will enter a bind sequence. In some cases, such as those when power is infrequently removed, this will not appear substantially different from what has been discussed under two-way button bind. In cases where power is cycled frequently, this method is either not acceptable or there will have to be some changes to prevent it from becoming intrusive to normal device operation. The likely change is to reduce the timeout as described above.

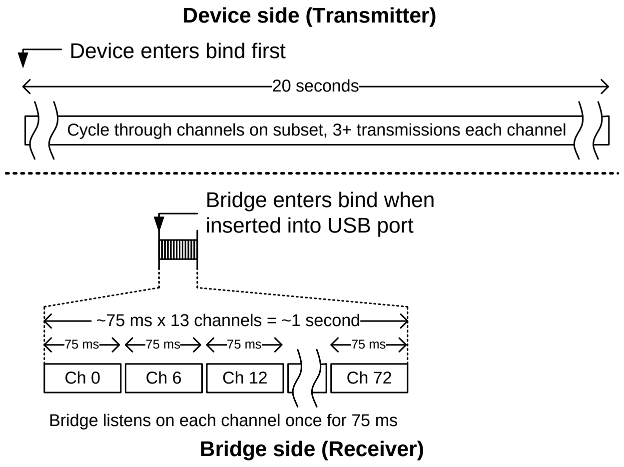
Power-up bind is typically implemented on one side only, with the other side using a button. Generally the timeout will also be kept small. It is possible to use power-up bind with a long timeout if power is rarely cycled. Examples are keyboard and mice for desktop users (Cypress battery life is typically a year or more), sensors in a building control network, or bridges built into some embedded systems.

Obviously, there are multiple combinations of power-up on one side or the other, or both, and short or long timeouts. The system architect must carefully consider the behavior that the user will encounter to determine if power-up bind choices are acceptable.

Example 1: Power-up bind on USB bridge; button bind on device

The bridge is USB based and will see frequent power cycles when it is removed from the PC, or when the PC shuts down or hibernates; therefore, it will have a short timeout of ~1 second. This is generally not noticeable when compared with the USB enumeration process. The device will use a standard button. To bind, the button on the device is pressed first, and then the bridge is inserted into the USB port. A bind channel subset and reduced PA are also used to generally increase robustness.

Figure 7. Sample Timing for Bridge Power-up Bind



Example 2: Power-up bind on building sensor; button on bridge

Sensors typically have batteries inserted once every few years. A long timeout on the device side is used so that the basic two-way button bind behavior is preserved (typical timing according to Figure 3). The long timeout will not be intrusive to the device operation. The user can start the bind process on either side: press the button on the bridge first and then insert batteries into the sensor, or the reverse. A bind channel subset is used for robustness, and a higher PA is used in case the bridge is a substantial distance away.

Example 3: Power-up bind on mouse/keyboard; button on bridge

This is somewhat less likely, but presents an opportunity to save cost by removing the buttons on the mouse and keyboard. Those that are not typically powered down can probably use the method described in Example 2. Others, particularly those intended for laptop users, have on/off switches to prevent inadvertent activation during transit. This power-up bind method employs a very brief bind sequence on power-up of the device (1 second or less) and a standard button on the bridge with a long timeout. For general use, the ~1 second timeout should be short enough that it is not inconvenient when powering up the product; however, it does slightly increase the power consumption of the device in cases where power may be cycled frequently. To bind devices, the bridge must initiate the bind process, then batteries are inserted into the mouse or keyboard, or the power switch is cycled. If both devices are being bound, the process is repeated for each one.

3.3.2 Advantage

- Using power-on to initiate bind on the device side can eliminate the need for one or more buttons, thus saving cost.

3.3.3 Disadvantages

- This process is not necessarily intuitive for the user; therefore, it needs to be described in user documentation (which users do not always read).
- There is a slight potential for increased technical support calls for users who do not understand the power-on behavior (Short timeouts require sequencing. Long timeouts make the device appear inoperable until the timeout is over).
- It potentially increases power consumption because the process repeats whenever power is cycled.

3.4 Traditional KISSBind

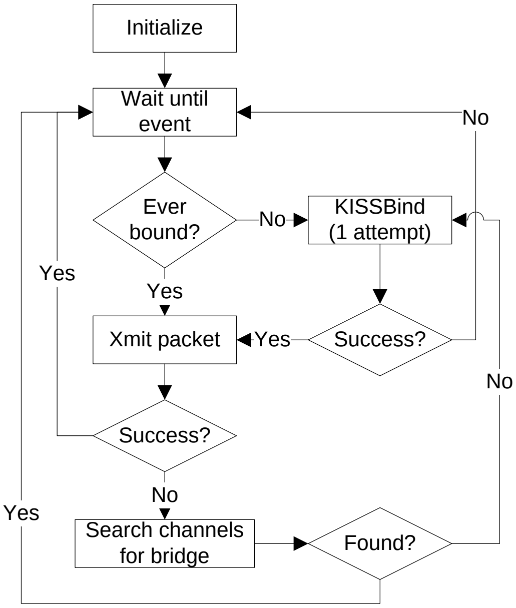
KISSBind is a method developed by Cypress to provide a simple dynamic bind method. It stands for “Keep It Simple Solution”, but is also activated by ‘kissing’ the units—bringing the device in close proximity to the bridge. Note that KISSBind does not necessarily replace other binding methods. It can coexist with one of the manual bind processes described above, thus providing enhanced functionality.

3.4.1 Description

There are a number of things about KISSBind that are very different from the bind methods described thus far. First of all, the bridge can support KISSBind when it is just listening for traffic as part of its normal operation.

On the device side, there are two situations where it enters a KISSBind mode. The first is when it is initially powered up without ever having been bound. The second is if it has repeatedly failed to get a response from its bridge. If it does not get acknowledgements and cannot locate the bridge on another channel, then it attempts one KISSBind sequence. If the attempt fails, it goes to sleep. When next awakened, it goes through the entire process again until it succeeds somewhere along the way.

Figure 8. Simplified Application KISSBind Flow

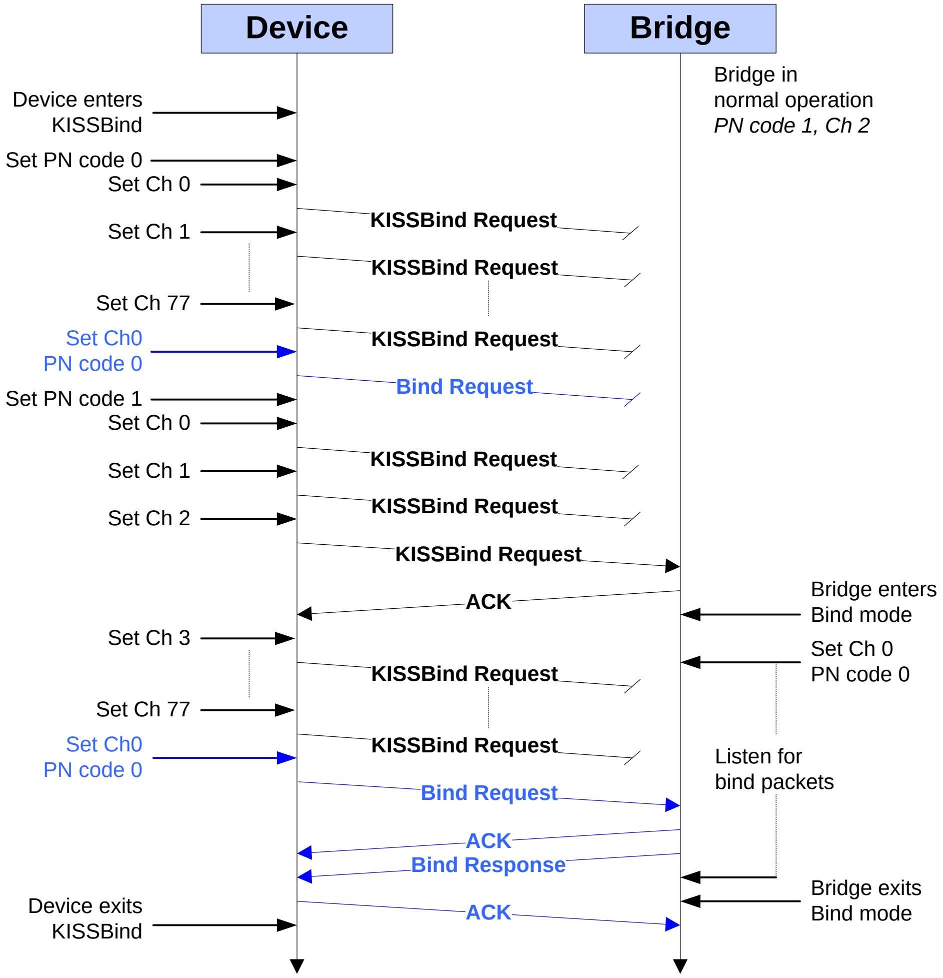


KISSBind also uses range, or more correctly it uses received signal strength, to determine when the device is ‘kissing’ the bridge. There are two sides to this. During transmission, the PA level is reduced. On the receive side the received signal strength indication (RSSI) is checked. If the RSSI value is not above a certain threshold, then the request is ignored because it is assumed that the device is too far away.

The bridge can use any channel and any PN code. Therefore, another unique thing about KISSBind is that the device must try all possible combinations of channel and PN code to locate a new bridge. Checksum seeds can also help differentiate systems; however, WirelessUSB LP allows checking both a seeded and zero seed CRC in hardware. Therefore, zero seed can be used for the KISSBind process.

For KISSBind, there are also two types of packets: a special KISSBind request and the bind request. The KISSBind requests are used by the device to search for a possible bridge on all channel and PN code combinations. It does not wait for a response to the KISSBind requests. Instead, it periodically goes to channel 0 PN code zero and transmits a bind request at reduced power. Bridges that have received a KISSBind request will automatically go to channel 0 PN code 0 to listen for bind requests. If the device and bridge find one another there, and satisfy the RSSI requirements, then the bind process completes.

Figure 9. Example KISSBind Sequence



3.4.2 Advantages

- KISSBind offers great flexibility by allowing devices to re-bind dynamically. Whenever an old connection is no longer active, a new connection can be established.
- It is conceivable that buttons can be removed from the devices, thus saving cost.
- Using transmit power and RSSI allows some degree of control by the developer. Vendors can choose if they will distinguish close devices or allow devices far away to bind using KISSBind.

3.4.3 Disadvantages

- This method is not intuitive and has an increased risk of technical support calls. Although conceptually simple—bring the devices close together and they bind—there is one key element that may not be obvious to the user. With battery-powered devices, the microcontroller is typically asleep whenever it is not needed. To attempt KISSBind requests, an action must be taken by the user to wake up the microcontroller and attempt communication. Therefore, it is not sufficient to just hold the device close to the bridge. While holding the device close to the bridge, the user must press keys, move the mouse, or take whatever action is necessary based on the product functions to make sure that it is awake. The user may not have any idea that this requirement exists and may not even know when the microcontroller is asleep or awake.
- Distance control is coarse, increasing the risk of cross-binding. Although the end user has a concept of distance, the actual device can only sense signal strength, as discussed in the range discrimination section earlier. It may be possible for variations of 2X, 3X, or more to exist in certain situations. What this means is that if the target range

for KISSBind was established at 12 inches, it may be possible to bind from 6 to 24 inches, or even 3 to 36 inches. Because KISSBind requests go out on every channel and PN code, this means that the bridge that responds is not the one that the user intended to bind with. For example, it is conceivable for my mouse to bind to the bridge of the person sitting next to me at a conference table. Calibration of RSSI may help to manage this, but there are multiple other parameters that are outside the control of the developer, which can still contribute to significant range variation. It may be possible to employ additional firmware methods to minimize or manage this scenario.

- If accidental binding does occur with an unintended bridge, it may be cumbersome to break. Once bound, normal range parameters are established, so communication could take place to 10 meters or further. The unintended bridge should be powered down, or the device and the intended bridge should be moved out of range of the unintended bridge.
- During development, the ideal received signal strength value is determined based on the range from the antenna on the device to the antenna on the bridge. When inside enclosures, the end user may not necessarily know where the antenna is located. This is especially problematic for keyboards or other large devices. If the developer is trying to control range very tightly, then it may be possible for the bind range to be shorter than the physical dimensions of the device. Thus, the user may try unsuccessfully to bind because in reality the antennas are too far apart even though the bridge and device are physically close together. This may be mitigated by using an icon, silk screen, or similar method to distinguish the location of the antenna on the exterior of the enclosure, but the user must still know what this means. Increasing the range to manage this might increase the risk of inadvertent binding described in the previous bullet.
- Due to the need to search all channels and PN codes, there is potential for the KISSBind process to take a long time. Based on existing Cypress implementations, it can take a few seconds to complete. In general this is not unacceptable, but there is a small risk that it will be longer than what a user expects, which can lead to confusion about what state the device is in.
- In current Cypress implementations, all of the actual bind requests take place on channel 0 and the default PN code. If there is substantial interference on this channel, then it may be possible for the bind to fail. There is no channel jumping algorithm. It may be possible to add such an algorithm but it will also increase the time for the process to complete, which could aggravate the condition described in the bullet above.
- When a bridge receives a KISSBind request, it stops normal operation and goes to channel 0, default PN code to listen for a bind request. It must stay on this channel for a brief period because the device may still be working its way through a list of PN codes. During this time, the bridge will not respond to devices that are already bound to it and attempting normal communication. This should be a very brief period, and should go unnoticed by the end user. This may be most significant if the device does not yet meet the RSSI requirements to bind with the new bridge. It could repeat periodically if the device attempting to KISSBind repeats the process multiple times. It is possible that the user may notice some small impact in performance.
- Power usage increases on the device due to the additional packets that are transmitted. Assuming communications are not lost on a normal basis and forcing the device into KISSBind mode, the impact should be very small.
- The code size is increased compared to other methods.

3.4.4 Derivatives

KISSBind on Bridge Only

One implementation that may help to control some of the less desirable behaviors of KISSBind is to create a hybrid where the KISSBind capability is only used on the bridge, but the device uses a button, or even a power-up sequence to initiate bind. This may be useful in situations where the bridge is not normally accessible, such as when integrated within a PC.

Instead of dynamically attempting KISSBind whenever communication fails, it would only be attempted upon a manual action initiated by the user. The bridge behaves as described above. The result is that this would become an infrequent action (maybe one time only), as opposed to the routine occurrence of the KISSBind described above.

Pressing a button (or powering up the device) takes care of the requirement to wake up the microcontroller. Since it is an infrequent operation that is manually initiated, it is probably acceptable to take a longer period of time, and even to implement an algorithm to try multiple channels, increasing the probability of success in the presence of interference. Increased power consumption is no longer a problem.

The concerns over range still exist, as well as the end user knowing where the antenna is, but it may not be necessary to control the range as tightly because binding takes place in a user controlled fashion. Pressing the button once more also provides a mechanism to break an inadvertent bind and re-attempt the process. Taking a bridge out of its normal operating mode to respond to a KISSBind request—even if it is not the one the user intends to bind with—is also manageable. The bridge that the user intends to bind with the behavior is expected when the button is pressed, and so acceptable. If it is an unintended bridge that is responding, recall that the interruption should be brief and almost unnoticeable, and since this is now an infrequent operation it should not happen again.

3.5 Out-of-Box KISSBind

Coming up with a suitable binding scheme for a specific application may involve mixing and matching some of the principles discussed in the first section of this application note, and even combining elements from some of the binding schemes.

As an example, consider a system binding solution called Out-of-Box KISSBind. The name KISSBind is kept because it preserves the “keep it simple” philosophy for the end user. ‘Out-of-Box’ is added because it is intended to be static—once bound it stays bound—as opposed to able to dynamically rebind on the fly with traditional KISSBind. Therefore, it will be used only for out-of-the-box binding when a device has never been bound before.

3.5.1 Description

It will be possible to bind multiple devices, but only one of any given type; for example, one mouse, one keyboard, and others. The bridge must keep track of when each type is bound, such as with a simple flag stored in flash, and will no longer accept KISSBinding from another device of the same type. On the device side it will only send out KISSBind requests if it has never been bound before. Otherwise, the same basic algorithm described in the previous section is still used. One possible relaxation is to ease the RSSI constraint to allow binding at slightly longer ranges of around a meter or so. This simplifies binding for the user because the risk of cross binding is small.

Theoretically it is possible to still experience cross binding, but that would require the user to power up the device for the first time in close range to two bridges that have never had that type of device bound before—a rare occurrence. Manufacturers may also want to provide a manual means to control the process, for example for technical support purposes. Therefore, a backup manual binding scheme is also implemented.

The backup binding scheme will basically be a modification/combination of power-up bind and button bind. The bridge will have power-up bind implemented as discussed previously—whenever the bridge is powered it will go into a brief (~1 second) backup bind listening interval. The device will have a derivative of power-up bind implemented. When the batteries are inserted and a specific button/key sequence is also followed, it will enter the backup bind mode. As an example, a mouse may require the left and right buttons to be held while the batteries are inserted to enter this mode.

The backup bind mode is similar to the power-up bind mode discussed previously, and shown in Figure 7. The device is placed in bind mode first and then the bridge is inserted into the USB port. If a new device is bound in this way, the bridge will disregard the flag indicating that a device of the same type has previously been bound.

3.5.2 Advantages

- Reduced cost due to elimination of buttons
- Simple user experience—user starts to use devices in close proximity to bridge and they will bind and function automatically—appears similar to a factory bound without the added manufacturing steps for the vendor
- Manual backup method allows control of the process and a means to fix cross-bound situations

3.5.3 Disadvantages

- Slight code size increase because multiple methods are employed
- Risk of cross binding still exists in rare cases
- Slightly increased power consumption, but only until bound
- Time to KISSbind is still slightly longer due to the need to search all channel and PN code combinations
- Bridges in other systems may be briefly interrupted—again, only until first bound

4 Summary

There are many options for binding devices in a wireless system. Button binding remains a very reliable method for managing this process. There are some creative solutions for providing more robust or low-cost binding implementations, but not all options are appropriate for all systems. Remember that the key question revolves around what type of user experience is intended. If this concept is kept in the forefront, it will help guide developers through the decision process.

This application note provided some key concepts and discussed multiple specific implementations, but it is far from all inclusive. Although these methods should solve most common problems, the basic concepts can also be used as a starting point to develop other solutions that are a good fit for specific applications.

4.1 Summary Table

Table 1. Binding Methods Summary

Method	Typical Use	Advantages	Disadvantages
Factory bind	Most systems	Easiest method with highest chance of successful user experience Low technical support call risk, low support cost No impact on power consumption in battery-powered systems Reduced code size if no alternative method	Risk of cross-binding on the manufacturing line Potential cost impact for added manufacturing steps Not suitable for systems where additional devices can be sold independent of the bridge
Two-way button bind	Most systems—where both sides are readily accessible	Simple system, good user experience Minimal risk of technical support calls Any order, leisurely timing requirement Modest power requirements Nominal code size	Small cost impact for button hardware Inadvertent button press ties up system for extended time New wireless users may not understand bind requirements
...Option for software initiation on bridge	Option for systems with robust user interfaces	Cost savings for removal of button	Not standard or obvious—requires user documentation, tech support call risk Development cost for host software
...Option for sequencing	Where inadvertent button press on one side may be possible and cause inconvenience	Reduced timeout—inadvertent bind button presses are imperceptible by user Bind power consumption on that side can be reduced	User must follow sequence or bind will fail Higher probability of bind failure—fewer opportunities to converge on channel
Power-up bind	When buttons are inconvenient or add too much cost	Cost savings for removal of button	Not standard or obvious—requires user documentation and technical support call risk Potentially increased power consumption
Traditional KISSBind	Systems that may need to be dynamically unbound and rebound	Dynamic unbind and rebind Potential cost savings by removing buttons (not recommended) Distinguish near and far devices	Not intuitive—technical support call risk Fine distance control is difficult, increased probability of cross-binding Difficult for the user to know which part of devices must be KISSed Longer (few seconds) time to bind Chance of bind failure due to interference if only one channel used for bind Other bridges may be briefly interrupted Slightly increased power consumption Larger code size

Method	Typical Use	Advantages	Disadvantages
Bridge-only KISSBind	Systems with bridges integrated (button not accessible)	Dynamic unbind and rebind Potential cost savings by button removal Distinguish near and far devices Reasonable power consumption Interruption of other bridges possible, but negligible impact	Fine distance control is difficult, increased probability of cross-binding Difficult for the user to know which part of devices must be KISSed Long (few seconds) time to bind Larger code size
Out-of-box KISSBind		Potential cost savings by button removal Simple user experience Manual backup when better control needed	Larger code size Small risk of cross binding Increased power consumption—but only until bound Long (few seconds) time to bind Other bridges may be briefly interrupted— but only until bound

Document History

Document Title: AN6066 - Wireless Binding Methodologies

Document Number: 001-15443

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1778266	KUH	11/27/2007	Re-catalogued application note.
*A	3711269	ZHC	08/13/2012	Template update. Changed abstract. Minor grammar and spelling corrections.
*B	4907169	LIP	09/04/2015	Template update.
*C	5836743	MALI	07/28/2017	Updated logo and copyright

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2007-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.