Overcoming UTMI Interface Limitations in USB-Enabled Handsets

Triton Hurd, Sr. Applications Engineer at Cypress Semiconductor
Ray Casey, Product Marketing Manager at Cypress Semiconductor

USB has quickly become widespread as the interface of choice for mobile handsets for connecting to a PC. This is due in part to the ubiquity of USB in the PC and laptop market – one would be hard pressed to find a new PC today that doesn't have at least a single USB port, with most providing two or four ports. Also, USB has become the standard method of transferring data to and from the PC – MP3 players (PMP), digital still cameras (DSC), flash drives, hard disk drives, etc. all employ USB as the transfer protocol of choice. As mobile handsets continue to integrate more and more features, such as higher resolution digital cameras, PMPs, and PDA functionality, users need a convenient method of transferring files to and from the handset. What could be better than the ubiquity and familiarity of USB?

With this in mind, handset processor vendors are beginning to integrate USB into the base band and applications processors. Today most processors integrate full-speed (FS) USB (12 Mbps), which is sufficient for small data transfer such as address book contact synchronization. With the addition of features such as MP3 players and high-resolution digital cameras, FS USB no longer provides sufficient throughput. Consumers are spoiled by the high-speed (HS) USB (480Mbps) transfers they enjoy with their dedicated PMPs and DSCs and are quickly disappointed by the experience of transferring MP3s and pictures to PCs using only their handsets' full-speed USB connection.

An example of the difference can be seen by comparing two commonly available handheld devices – the first supporting high-speed and the second supporting full-speed USB – transferring a fixed amount of data from a host PC to each device, and recording the time it takes for the transfer to complete. It takes a high-speed USB device approximately 33 seconds to transfer 105 MB of data from the host PC to the handheld device. The full-speed USB device takes almost 13 minutes to perform the same transfer! With cutting-edge flash-based handheld devices currently supporting 8GB of data storage, it could take over 17 hours to update files using full-speed, versus 44 minutes using high-speed USB. Hard disk drive (HDD) based handheld devices support up to 80GB, requiring an increase of 10 times to 170 hours using full-speed versus 7.3 hours (440 minutes) using high-speed USB. While users may not want to upload or download this amount of data every time the handheld device is connected to the PC, the example serves to illustrate the disparate consumer experience when using full-speed versus high-speed USB.

Handset processor vendors are reacting to HS USB's pull and are integrating HS USB directly on chip. However, they are integrating USB only as far as the Serial Interface Engine (SIE). While FS USB can be realized in digital-only circuitry, HS

USB achieves a much higher data rate (40x) and therefore requires an analog PHY. As handset processor vendors continue to cut costs by moving down the technology geometry curve, they run into problems integrating the analog PHY because analog circuitry does not scale as well as digital with each shrinking technology node. Processor vendors are therefore choosing to omit the PHY and exposing standard interfaces such as UTMI (USB 2.0 Transceiver Macrocell Inteface) and ULPI (UTMI+ Low Pin Interface) for use with standard USB PHYs.

The UTMI interface consists of between 22 and 26 pins, depending on whether an 8- or 16-bit data bus is implemented. This translates to a requirement for 22 or 26 processor GPIOs when considering all data and control signals. Although newer generations of processors are increasing the number of available GPIOs, the requirements on GPIOs to support an ever increasing feature set far outweighs the number of GPIOs available. This forces designers to pick and choose the features they need to add, or come up with creative ways to reuse these limited GPIOs.

Examples of other functions which typically need GPIOs to interface to the processor include:

- Camera Control Module
  This module takes in data from an image sensor and transfers it to the available storage media (SD, NAND, HD, etc.) via the processor. This is the most common function which is shared with the UTMI transceiver because, from a usage model, the user will not likely need to use the camera while transferring files to and from the PC.
- DVB-H (Digital Video Broadcast – Handheld)
  This device takes in DVB signals and transfers them to the processor so that users can watch live television on their mobile handset LCD screen. Again most likely users will not need to transfer files while watching television.
- Bluetooth
  Bluetooth modules handle all communication with Bluetooth-enabled peripherals and radios. Bluetooth is a technology which is “always on” or always searching for peripherals, so it is unlikely that this will be shared with the UTMI transceiver.

Overcoming UTMI Limitations

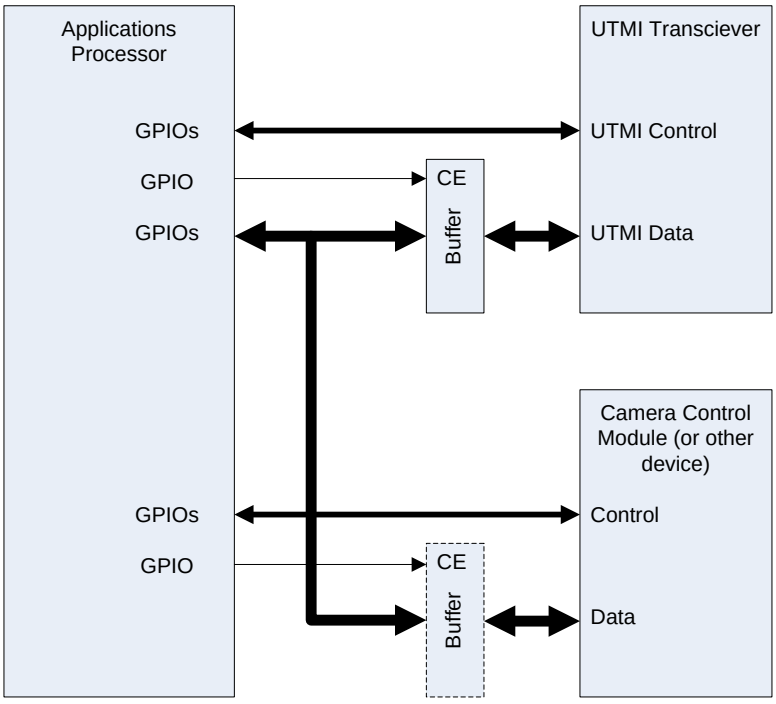
There are two basic approaches to reusing GPIOs on a processor. The first is to isolate each device using external tri-state buffers, allowing each path to be turned on individually by enabling one path and disabling the other via the relevant tri-state buffer device. The second approach is to put the peripheral devices into a mode which tri-states the interface, freeing it up for another function.

The first approach requires additional external components. In handset designs, board space is at a premium and when designers have the opportunity to cut down on component count, they take it. Eliminating external tri-state buffers drops BOM costs, board area requirements, and routing complexity. Said another way, designers may go so far as to omit certain features rather than add extra components to enable those features.

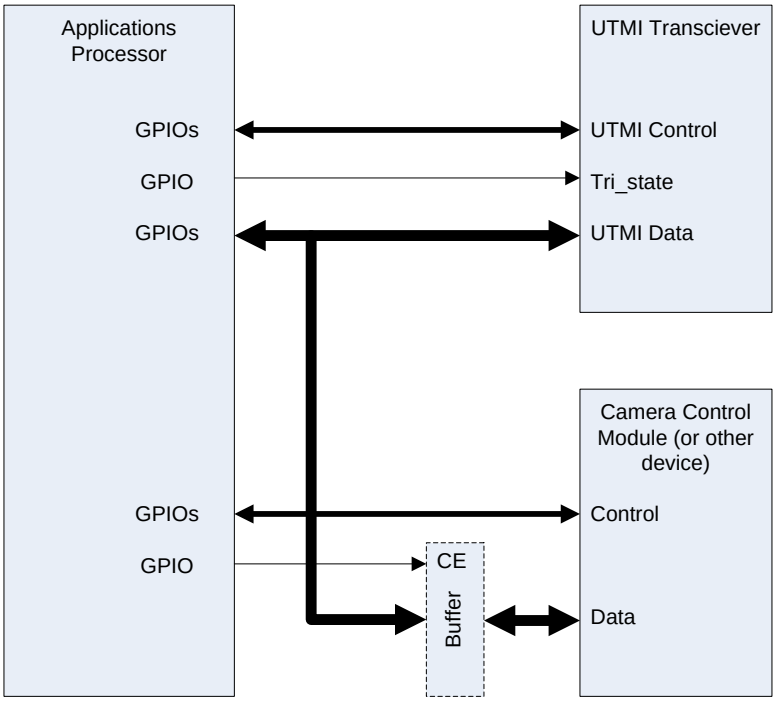
Not only do mobile handset designers frown on the requirement of additional components required by the first solution, they will also run into design complications in regards to timing with the addition of propagation delay through the tri-state buffer. For example, when designing with a UTMI interface in 8-bit data bus mode, per specification the interface is clocked at 60MHz. This provides approximately 16ns of timing budget. Good transceivers will have a setup time of 4ns, while the clock out time of the processor can be up to 8 ns. This leaves 4ns of the timing budget. If a buffer is inserted into the path it will have a propagation delay of 4 to 6 ns leaving no timing budget for path delays and margin.

Due to added cost, complexity, and challenges associated with the first approach, designers have sought out an alternative manner of putting the peripheral devices into a mode which tri-states the interface, freeing it up for another function. This in essence is the integration of tri-state buffer functionality into the peripheral device directly, thereby removing the need for the external component. With this approach designers can successfully reuse GPIOs while keeping component count at a minimum. Figure 1 illustrates both of these methods.

GPIO Sharing Without Tri-State Mode



GPIO Sharing With Tri-State Mode



Buffer

NOTE: This buffer is only needed if the Camera Control Module (or other device) cannot tri-state its data bus.

Figure 1: GPIO Sharing Methods

While this second option eliminates the need for external tri-state buffers, it too is not without its complications, the most obvious of which is ensuring that the UTMI Transceiver is able to tri-state the UTMI interface. This may seem trivial, but the UTMI specification was not developed with the intention of sharing GPIO pins and is defined to always be driving or ON. As a consequence, today's UTMI transceivers need to integrate the capability of tri-stating the UTMI interface.

Another pitfall is from the system level. There are only specific times when it is safe to tri-state the UTMI interface. It may seem that the obvious answer is when there is no data being transferred on the interface, but the mobile handset (when not acting as an On-The-Go (OTG) host) is a slave and waits for the USB host to initiate a transfer. How is the processor to know when the USB host is going to request a data transfer if the UTMI interface is tri-stated? There is only one safe time to tri-state the UTMI interface on the transceiver and that is when the transceiver is put into suspend mode by the processor because there is no cable USB connected to the host. This is the only time when it is guaranteed that there will not be any transfers which have been initiated by the host.

There are two types of suspend. One is issued by the USB host, called a USB suspend, and another is issued by the processor, called a device suspend. During a USB suspend, the transceiver cannot be put into suspend mode because the processor must monitor the UTMI Line State pins for a host issued USB resume, indicating future transfers. If at any point the USB cable is removed, there would no longer be a connection to the host and the processor may put the transceiver into a device suspend. The processor will know when the USB cable has been removed because it monitors Vbus, which is a voltage applied from the host that is only present when there is a connection to the host.

The following is an example of a typical usage model, and Figure 2 illustrates all of the states in this example. When the handheld device is not plugged into a USB host, the transceiver is put into device suspend mode by the processor via the suspend pin. The UTMI interface is then tri-stated using the tri-state mode enable pin and the associated GPIOs are freed up to be used by other functions. It remains in this state until the user decides he/she would like to sync the mobile handset to a PC to transfer emails, or to perform a mass storage transfer of pictures or MP3 files. He/she plugs a USB cable into the mobile handset and then into the PC and thus a connection to the USB host is made and Vbus is applied. The Vbus detection circuitry will notify the processor that a valid Vbus is present and the applications processor will disable any other devices or functions which are sharing the UTMI interface, disable tri-state mode, and disable the transceiver suspend (effectively enabling the transceiver). After the mobile handset has been updated, if the user leaves it connected to the host, it may decide to issue a USB suspend to the device because it does not need to use it. If the host issues a USB suspend, the transceiver cannot be put into device suspend mode because the state of the D+/D- lines will need to be monitored for

a USB resume from the host. The only way to do this is for the processor to monitor the Line State UTMI signals on the UTMI interface of the transceiver. If at any point the user decides to unplug the USB cable, the processor will detect that Vbus is no longer present and the transceiver may be placed into suspend and the UTMI interface tri-stated again.

Reusing GPIOs provides a cost-effective means for designers of portable consumer electronics to provide the HS USB functionality required to meet the base expectations of today's users. By integrating tri-state capabilities on UTMI PHYs, it is possible to avoid introducing external components that not only increase BOM costs but also increase design size. GPIO reuse allows designers to preserve other device features, resulting in a feature rich handset design that saves on both cost and board space while best meeting customer needs.

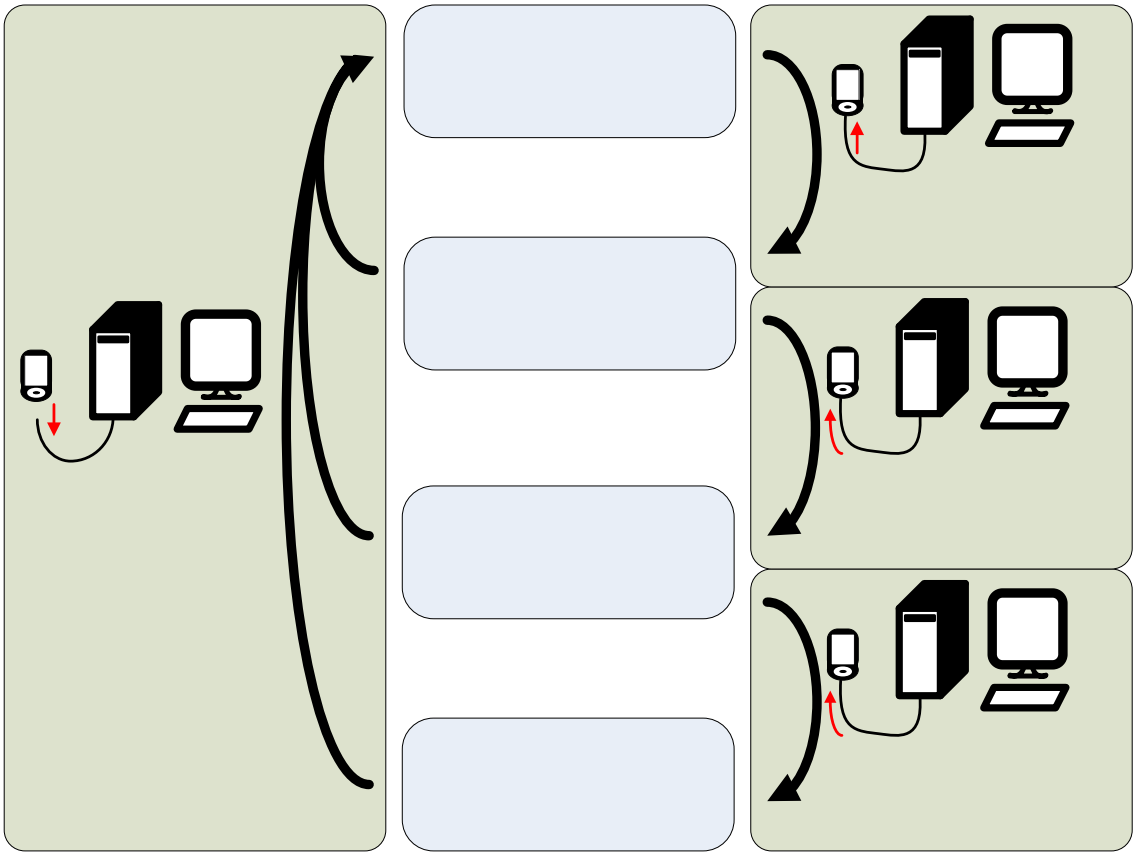


Figure 2: Usage Model Flow Chart

State 1: No Conn
Transceiver St
UTMI Interfac