MESSAGE SIGNALLED INTERRUPTS IN MIXED-MASTER CONTROL

W. W. Terpstra, GSI, Darmstadt, Germany
M. Kreider, GSI, Darmstadt, Germany

Abstract

Timing Receivers in the FAIR control system are a complex composition of multiple bus-connected components. The bus is composed of Wishbone crossbars which connect master devices to their controlled slaves. These crossbars are in turn connected in master-slave relationships forming a DAG where source nodes are masters, interior nodes are crossbars, and terminal nodes are slaves. In current designs, masters may be found at multiple levels in the composed bus. Bus masters range from embedded microcontrollers to bridges from PCIe, VME, USB, or the network.

In such a system, delivery of interrupts from controlled slaves to masters is non-trivial. The master may reside multiple levels up the hierarchy. In the case of network control, the master may be kilometres of fibre away. Our approach is to use message signalled interrupts (MSI). This is especially important as a particular slave may be controlled by different masters depending on the use-case. MSI allows the routing of interrupts via the same topology used in master-slave control. This paper explores the benefits, disadvantages, and challenges uncovered by our current implementation.

INTRODUCTION

To coordinate accelerator activities at GSI, we need a hard real-time control system. The hard real-time components of the control system are implemented using logic chips whose wiring can be reprogrammed in the field, called FPGAs. This makes it possible for our existing hardware to accommodate some of the changing requirements that arise as physicists revise their research goals. It also means that we have a great deal of flexibility in terms of how we build and connect the hard real-time components inside the FPGA.

For timing receivers, we have standardized on the pipelined Wishbone B.4 bus standard [1]. By using this open standard inside our FPGAs, we can interface components of our design more readily with components from CERN and the open hardware community at large. Our experience thus far, about six years, has shown that Wishbone strikes a good balance between simplicity and flexibility. Simple components can easily speak Wishbone, but the bus scales well enough that today our FPGA designs include nearly fifty distinct Wishbone-connected components.

In a complex system like ours, many components must communicate with each other. Most often, this communication flows from logic components, which direct the planned accelerator behaviour, to physical interfaces, which perform the external signalling to control magnets, measurement equipment, and displays. However, sometimes this communication flow must be reversed. If a magnet power supply encounters an error condition, it must be able to notify the components which control the system. The Wishbone standard does not include such a mechanism, and a traditional interrupt-based system is inadequate for our needs.

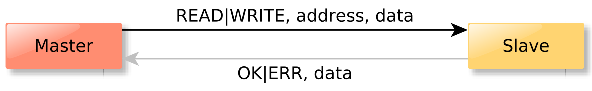

Figure 1: Wishbone connects one master to one slave. Masters initiate requests. Slaves respond with success or failure.

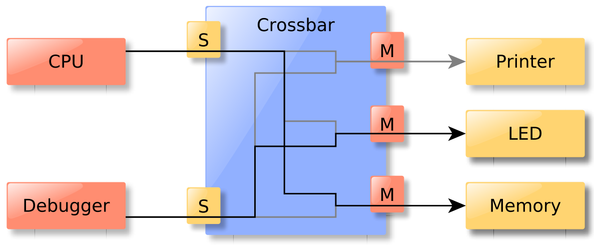

Figure 2: A crossbar providing two slave ports and three master ports. It internally connects each master to one slave.

This paper will explain how we have extended Wishbone to include a notification facility, based on message signalled interrupts (MSI). Our approach essentially mirrors the Wishbone bus to allow messages to flow bidirectionally throughout the design. While this approach carries an increased interconnect cost, it is very flexible and requires only a minor revision to the Self-Describing Bus standard [2].

WISHBONE BASICS

Wishbone is a master-slave point-to-point bus protocol. This means that it describes how to connect a single master to a single slave, as shown in Figure 1. All communication is initiated by the master, which sends bus operations (reads or writes) to the slave. The slave then executes the requested operation and reports any resulting data and the success or failure of the operation.

Bus operations include an address. Typically, the slave uses the address to determine what to do with the operation. For a slave that implements memory, the address is simply the location in its table where the value should be read or written. However, for most slaves, the address acts as an indication of the type of command. Writing to a particular address might cause the magnet to instantaneously change its field strength. Writing to a different address might cause the magnet to begin slowly ramping the field from one strength to another. Reading from one address might report the current field strength, while another address could report current or voltage. In any case, Wishbone slaves have a fixed number of addresses that a master can read or write.

Pre-Press Release 23-Oct-2015 11:00
Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-148-9

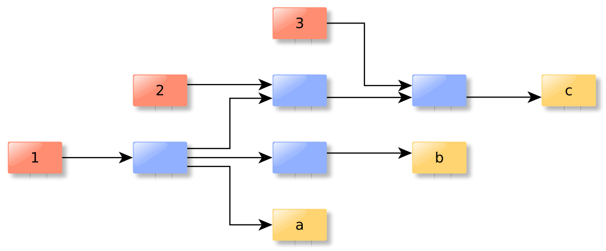

Figure 3: Real systems often include many nested crossbars.

While a point-to-point protocol is already useful for combining two components, sophisticated systems generally need to connect more than two components together. This is the role of a Wishbone crossbar switch, as illustrated in Figure 2. The crossbar implements several slave ports, suitable for connecting Wishbone masters, and several master ports, suitable for connecting slaves. Inside the crossbar are wires which allow each master to send messages to each slave.

To decide which slave to connect to a master, the crossbar uses the Wishbone address. Recall that each slave has a fixed number of addresses. The crossbar essentially concatenates the slave addresses together to create one giant slave device. The address in the first bus operation from a master is used by the crossbar to decide which slave to connect to the master. Thereafter, the master stays connected to that slave until their communication is complete. If a different master wants to access the same slave, it must wait. However, as shown in Figure 2, two different masters can communicate with two different slaves, without waiting.

A single crossbar is already enough to build quite complicated systems. However, as the design grows in complexity, multiple crossbars start to appear. Consider, for example, a host motherboard with slots for inserting addon cards. The inserted cards might contain several slaves, connected together by a card crossbar. The motherboard hosting the slots likely also has some masters and slaves, connected by a host crossbar. When the card is inserted, the masters in the hosting motherboard would like to access the slaves in the addon card. This is achieved by connecting a master port of the host crossbar to a slave port of the card crossbar. Now the host masters have access to the host slaves and, via the crossbar-crossbar connection, the card slaves.

Beyond the physical need for nested crossbars in the previous example, nesting crossbars is also a useful organizational tool. For example, CERN provides a small White Rabbit (WR) crossbar with masters and slaves that cooperate to precisely synchronize clocks [3]. We use this WR design as a component in our larger system. Treating their entire WR crossbar conceptually as a single slave component on our larger bus allows us to cleanly separate our code from theirs.

In any case, once you have multiple nested crossbars, you end up with a Directed Acyclic Graph (DAG) like in Figure 3. As shown, we have masters 1, 2, and 3 connected to slaves *a*, *b*, and *c*. Of course, a real system would probably include far more masters and slaves on each crossbar, but we keep the

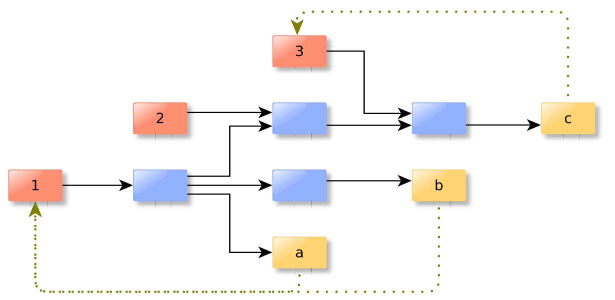

Figure 4: Interrupt lines connect a slave to one fixed master.

example simple for illustration. Notice that masters 2 and 3 only have access to slave *c*, whereas master 1 has access to all three slaves. This is typical for a complex Wishbone bus. When well organized, the restricted access of masters to the bus helps us reason about the system as a whole.

THE INADEQUACY OF INTERRUPTS

As already explained, only masters initiate the exchange of messages in the Wishbone bus. If a slave has information for a master, it must wait for the master to read it. Unfortunately, this model is too simplistic. Consider a slave which rarely has anything to tell its master, but sometimes it has an urgent message that must be delivered immediately. In this case, the master must wastefully read the slave repeatedly, forever. This is not always feasible, so we need a way for slaves to signal to masters that they need service.

The classic approach is hardwired interrupt lines, as illustrated in Figure 4. In this model, the slave has a single wire (the interrupt line) with which it can signal its need for service. During normal operation, when it has nothing to be read out, it leaves the interrupt deasserted. Once it has something to read, it asserts the interrupt line. This informs the master that it must immediately read the slave. Once the data has been read out, the slave deasserts the interrupt.

The advantage of this approach is that it is very inexpensive. It only requires a single wire from slave to master. In an FPGA, this benefit is not really important, but on a PCB this can be a decisive factor. However, while cheap, this approach has serious shortcomings.

Firstly, an interrupt line typically connects one slave to one master. In Figure 4, slave *c* was connected to master 3. It might well be that master 3 is the only master which ever controls slave *c*, in which case this is fine. However, in our designs, it is often the case that a slave might be controlled via different masters depending on the deployment. For example, we might control a timing receiver via a PCIe slot in a PC. The same card might also be controlled over the network or via USB. Depending on how the card is connected, the interrupts need to go to different masters. This is not possible in a simple interrupt line scheme.

Secondly, there are usually only a fixed number of interrupt lines. In Figure 4, both slaves *a* and *b* are connected to the same interrupt line of master 1. When the master sees

Pre-Press Release 23-Oct-2015 11:00
Copyright © 2015 CC-BY-3.0 and by the respective authors

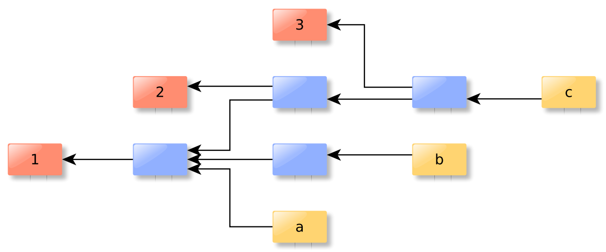

Figure 5: MSI reverses Wishbone, allowing each slave to send notifications to any of the masters which can control it.

an asserted interrupt line, he must check all slaves attached to that interrupt line for the source of the interrupt. If a master does not have support for the slave which asserted an interrupt, it is also unable to service the interrupt. Thus, the interrupt stays asserted and all other slaves sharing the interrupt line are now unable to signal the master.

Thirdly, a single slave might have several reasons it asserts an interrupt. As the interrupt line is only a single bit, it cannot indicate the reason. Different reasons might have different priorities. The master cannot know how urgent the interrupt was until after it has checked all possible reasons for the interrupt. In a bus with forty slaves, each with multiple reasons for an interrupt, it could take a long time to determine that the interrupt was not urgent after all.

Finally, interrupt lines break composability. We want to freely nest crossbars for code reuse and clarity. Interrupt lines cannot be plugged together in the same way we connect crossbars. We would like to plug two crossbars together and suddenly have the upper masters be granted access to the nested slaves. However, the interrupt lines of those slaves are probably already connected to the masters at their own crossbar. This means that the new outer masters may be unable to use the slaves properly.

MESSAGE SIGNALLED INTERRUPTS

One alternative to interrupt lines is Message Signalled Interrupts (MSI). In this scheme, we allow slaves to send messages towards masters. In a very real sense, this flips the role of masters and slaves; see Figure 5. Wherever there was a crossbar, it is now the slaves which initiate the message and the masters which receive them. In our system design, we implement this using a second Wishbone bus.

On the MSI Wishbone bus, masters are now essentially Wishbone slaves. For clarity, we will always denote master versus slave from the point-of-view of the regular Wishbone bus. Since masters now receive MSI Wishbone messages, they should provide a small address space. Each address in this space is analogous to a distinct interrupt line. However, 16 address bits suffice for 64K interrupt lines, enough for most uses. The MSI crossbars route messages from slaves to masters just like the normal crossbars from master to slave.

For each type of notification they will send, slaves should provide a configuration register to set the MSI target address. They should also provide an enable register that indicates if that type of notification should be sent at all. When a master wants to receive MSIs, he puts his own address into this target address register and enables the notification. When the slave wants to notify the master, if this notification is enabled, he sends a MSI out the MSI Wishbone bus to the configured MSI target address.

The data field of the MSI can be use for parameters within a notification. For example, consider a slave with a FIFO. It should provide three types of MSI notifications, each with its own configurable MSI target address: empty, full, and level. The empty notification has either a 0 or 1 in its data field and is sent when the FIFO changes between empty and not empty. Likewise, the full notification sends a 0 or 1. If enabled, the fill notification would be sent whenever the level changes, indicating the new level in the data field.

Compared to interrupt lines, this approach resolves all of our concerns. Firstly, any master which can control a slave can receive MSIs from that slave; the bus topology has all arrows between them reversed. Secondly, the masters have many thousands of MSI addresses, and can instruct distinct slaves to use distinct MSI addresses. Thus, a master immediately knows which slave sent the message based on the target address. Thirdly, if a master wants to distinguish messages by priority, he can provide different queues on different addresses. Then he instructs a slave to send urgent messages to one queue and unimportant messages to another queue. For example, he could connect a slave FIFO's full notification to an urgent MSI queue, disable the empty notification, and connect the fill notification to an unimportant queue. Finally, the MSI approach is just as composable as Wishbone, because it *is* Wishbone.

Clearly, this approach could double the interconnect cost in the worst-case. However, slaves which do not generate MSIs can be omitted, as can masters which do not receive MSIs. Furthermore, MSI bus logic tends to be simpler on both sides, because only writes are allowed. Nevertheless, there is an area price to be paid for this approach. We consider this an acceptable trade-off inside an FPGA.

As an optimization, when Wishbone leaves the FPGA, we sometimes use interrupt lines behind the scenes. In this case, a cheap interrupt-based external protocol is used to tunnel Wishbone and MSI messages. However, this use of interrupt lines is invisible to Wishbone masters and slaves. It is an implementation detail of the Wishbone bridge.

ADDRESSING AND SELF-DESCRIPTION

One detail we have ignored till now in Wishbone is how crossbars compose addresses recursively. The general idea is that a Wishbone slave should ignore the high bits it does not use itself. Thus, the GPIO in Figure 6 only pays attention to the low 5 address bits. The crossbar which connects it and the LED, inspects the 6th address bit to decide if an operation goes to the LED or the GPIO. This means that when accessed via this crossbar, the GPIO register 0x4 has address 0x24 outside the crossbar. Similarly, from the CPU's point-of-view, that GPIO register has address 0x124.

Pre-Press Release 23-Oct-2015 11:00
Copyright © 2015 CC-BY-3.0 and by the respective authors

ISBN 978-3-95450-148-9

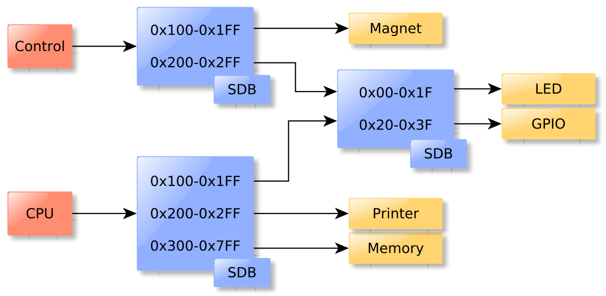

Figure 6: Wishbone routing is recursive. Slaves and crossbars ignore the high address bits matched earlier on the path.

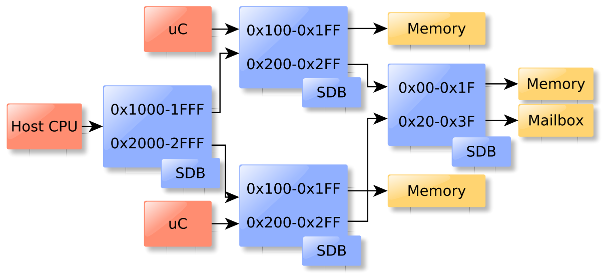

Figure 7: With shared bus resources, a master (Host CPU) sometimes sees a slave (Mailbox) twice under two addresses.

However, this compositional approach leads to something a bit surprising. That same GPIO register has address 0x224 from the point-of-view of the Control master. The correct way to understand this is that **an address describes a path**, not a destination. The address 0x224 tells you how to get from the Control master to register 0x4 of the GPIO.

If addresses do not describe destinations, how do we know what is on the bus? This answer is provided by Self-Describing Bus (SDB) records [2]. Each crossbar includes a small table that describes what is attached to it at each address range. In a sense, SDB describes the outgoing arrows on each crossbar. A Wishbone master recursively reads these SDB records, exploring all paths through the bus and composing their associated addresses.

When a master finds two identical devices via SDB exploration, it presumably wants to tell them apart. This is the job of the device driver. SDB will tell you that there are two paths to LED devices. The driver must query the LED slaves to discover that one LED slave should bink to indicate network activity and the other indicates power.

The final wrinkle, which also affects MSI, can be seen in Figure 7. Here, we have two microcontrollers (μC), each with their own private memory and a shared mailbox. However, there is a supervisor Host CPU which has access to both of their private memories. Diamond patterns like this become unavoidable in larger designs. In this example, it is clear that the Host has two paths to the mailbox. Armed with our refined understanding of addresses, it is obvious that this means that mailbox register 0x8 has two addresses from the point-of-view of the Host, 0x1228 and 0x2228. There are two paths, and so two addresses. As with identical LEDs, it is the mailbox driver's job to recognize that the two addresses are just different paths to the same device.

FINDING YOUR OWN NAME

Since the MSI bus is just the normal Wishbone bus inverted, it should be no surprise that masters have different addresses when viewed from different slaves. Again, addresses describe a path, not a destination.

When a master wants to receive an MSI from a slave, he must configure the slave's MSI target address register with his own address. However, the master does not have a unique address. He must use the address which describes the path from the slave back to himself.

We have seen that masters recursively scan SDB records to locate all paths through the bus. As the master explores paths, he constructs the corresponding path addresses top-down. However, on the MSI bus, the master is the target. For this reason, a master which scans SDB should simultaneously construct the backwards path address from that location on the bus to himself. This MSI path address is constructed bottom-up. For each path found during SDB bus enumeration, the master has a triple of information: the SDB meta-data describing the terminal slave, the address describing the path to that slave, and the address describing the path from that slave back to the master.

Unfortunately, to construct the bottom-up address path from slaves requires information about the masters in SDB. These new records have not yet been standardized.

CONCLUSION

Wishbone scales well from simple to very complicated bus systems. To support slave-to-master messaging in Wishbone, we take a MSI approach. This gives us great flexibility at the cost of increased interconnect area. In particular, we can combine crossbars together hierarchically while retaining the ability to deliver notifications from slaves to all the masters which might control them. To realize our design, it is necessary to revise SDB [2] so that masters may discover their own addresses when configuring slaves for MSI.

REFERENCES

[1] R. Herveille. WISHBONE System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores - Revision B.4, 2010. URL `http://opencores.org/opencores,wishbone`.

[2] A. Rubini, W. W. Terpstra, and M. Vanga. Self-Describing Bus (SDB) Specification for Logic Cores - Version 1.1, April 2013. URL `http://www.ohwr.org/projects/sdb`.

[3] P. Moreira, J. Serrano, T. Wlostowski, P. Loschmidt, and G. Gaderer. White Rabbit: Sub-Nanosecond Timing Distribution over Ethernet. In *International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, 2009.

Pre-Press Release 23-Oct-2015 11:00

Copyright © 2015 CC-BY-3.0 and by the respective authors