Block Interconnection

Today’s topics divide into two:

Problems of logical interfacing

- Standardization of interfaces
  - AMBA
  - OCP
  - ...
- Interface operation

Problems of electrical interfacing

- Skew
  - clock skew source/dest.
  - bit skew in buses
- Round trip delays
- Drive capacity

The latter set of topics is more in the area of the later lectures.

Intellectual Property (IP)

In any sizeable SoC it is likely that you will not build everything yourself. Various blocks of ‘Intellectual Property’ (IP) are available, freely or at a price, for incorporation into other designs. This is the business model of, for example, ARM Ltd; they do not make chips themselves but license processors and other designs to Apple, Nokia, … These equipment designers will purchase rights to use IP blocks from multiple sources and integrate those with their own application-specific logic to make a purpose built chip.

IP blocks typically come as ‘black boxes’ and it is the function and the interface which are of interest to the developer. Having some standard interfaces allows blocks to be composed easily.

Advanced Microcontroller Bus Architecture (AMBA)

AMBA is an open standard (or set of standards) which have become a de facto standard for on chip interconnect. The standards specify the list of signals used in the interconnection and their timing relationship on a cycle-by-cycle basis. It was first introduced by ARM Ltd in the 1990s and has been developed continually, since.

AMBA comes in several ‘flavours’, including:

- Advanced Peripheral Bus (APB)
- Advanced High-performance Bus (AHB)
- Advanced eXtensible Interface (AXI)

which are used here as examples.

The different standards represent different points in the complexity/performance spectrum. Thus APB is simple but slow – intended for communication with many, low-bandwidth peripheral devices. Because peripheral accesses are rare in comparison with memory reads and writes a few slow cycles do not impact overall performance significantly.

AXI is better suited for high-bandwidth communications. An example would be the data bus from a memory controller which was frequently used. It allows bursts of data to be communicated and several outstanding transactions at any time, so operations can be pipelined. The price is a significant increase in complexity at the interfaces.

Open Core Protocol (OCP)

AMBA is not the only standard in use for on-chip interconnect. OCP is an attempt to provide a standardised ‘socket’ for plugging together Intellectual Property (IP) blocks to make a chip.

‘Traditional’ bus

Example

Address

Read

Read data

Write

Write data

- Asynchronous bus (timed by strobes from master)
  - Timing generated by clocked circuit but no clock on the bus
- Everything happens successively during cycle
- Cycle may be extended with ‘wait’ states

As seen in lab. on:
- ARM-FPGA interface
- Framestore SRAM

Asynchronous bus

The example in the slide is not the only protocol for bus timing. Another common approach uses an enable (CE) and a direction signal to specify the operation.

Although the data is shown here as unidirectional, off chip buses typically use bidirectional data signals so must be either reading or writing when active. This is due to pin restriction on the package (and wiring on the PCB).

On chip buses are limited by distance but not (particularly) restricted by width because there is a considerable wiring resource on a chip. However on-chip signals are now ‘universally’ unidirectional so that electrical buffers can be inserted along the wire to keep switching edge speed reasonably rapid.

Wiring

Let’s elaborate on that point.

A simple model of ‘load’ on a gate estimates a ‘lumped’ capacitance. 200 µm of wire will have twice the capacitance of 100 µm.

Assuming the same driver, the edge time on the longer wire will be correspondingly greater; thus the signal delay will be greater.

Possible solutions:
- Increase the drive strength
- Decrease the load

However the wires also have resistance which slows down the edge more at greater distances from the driver. The first solution is therefore not as effective as might be though at driving longer wires.

The load can be decreased by ‘cutting’ the wire and inserting buffers (amplifiers) at intervals. These also insert delay but keep the edges fast.

Buffers have an input and an output so the wires, necessarily, are unidirectional.

Buffers

The term “buffer” as applied here refers purely to an electrical amplifier. “Buffer” is also used to refer to, for example, latches which hold data and are thus part of the logic. Beware of potential confusion!
Advanced Peripheral Bus (APB)

APB is basically a straightforward microprocessor bus. The bus master puts out a command, address and (possibly) write data or (possibly) latches read data at the end of the cycle.

- Simple
- Single master
- Used for low speed peripherals

The first APB spec. performed every transfer unconditionally in two clock cycles. This was subsequently extended so that slow peripherals can insert extra ‘wait’ states to extend the cycle time if they cannot respond quickly enough. Wait states may be acceptable when communicating with peripheral devices because such accesses are infrequent so the penalty is small.

Another extension was an error signal, so the failure (abort) of a bus cycle can be signalled.

APB

Bus Errors

When a bus master is designed it is not always determined what will be on the other end of it.

- On a ‘motherboard’ different ‘expansion cards’ may be inserted
- On a SoC the hardware is fixed – but SoC designs may differ and the designer may not want to customise the master each time

A bus transaction may be successful – or it may fail for a number of reasons:

- Segmentation fault: the address is illegal for that process at that time:
  - Outside the allowable range for that process.
  - Writing to a ‘read’ only area.
  - User access to a privileged (operating system) address.
- Page fault: the address is legal but there is no physical memory present at the time.

Segmentation faults are typically ‘fatal’ for a thread; page faults require some rearrangement of the memory map.

Many of these are detected by a Memory Management Unit (MMU) before reaching the bus. However, some requests are not, or cannot be, trapped there and cause a bus cycle.

A bus will typically have a status signal returned from the slave device which indicates whether the cycle has completed successfully or if there was a bus error and it has been aborted.

- On a read bus error any returned ‘data’ will be invalid
- On a write bus error the write did not complete

Example

A peripheral I/O device may have only a small number of registers (say 16) but be allocated a ‘page’ (say 1KB) of the memory map. It could indicate if an access was apparently to that device but not to one of its valid registers. Alternatively, it could indicate an attempt to write to a read-only register.

This cannot be done by a typical MMU which will not resolve translations to individual words, only pages.
Advanced High-performance Bus (AHB)

AHB is a pipelined bus intended to perform one transfer per clock cycle.

- Moderately complex
- Multi-master via centralised arbitration
- Bus cycles can be extended or aborted
- Used for processor buses on medium performance devices (e.g. ARM9)

AHB increases performance by pipelining. For example, in a read operation it outputs an address and status asking for the read on a rising clock edge. This is decoded and selects the appropriate slave device.

On the next active clock edge the slave is expected to latch the address and start the read. At this point the bus master can start the next cycle.

On the next active clock edge the master must:
- latch the first input data
- provide output data if the second cycle was a write operation
- start the third cycle (if appropriate)

This sequencing allows faster bus throughput but causes certain difficulties when things don’t go smoothly.

- If a peripheral is slow and needs to insert wait states it does this in the data phase. Other peripherals need to monitor this because, if one is being addressed ‘next’ it needs to defer starting.
- If a bus cycle is to abort the ‘pipeline’ needs to be ‘flushed’. All slave devices must watch for other devices aborting so they don’t start the subsequent cycle, which may already be being requested.

AHB operation is pipelined, so that as one set of data is transferred the subsequent address can be sent.
Advanced eXtensible Interface (AXI)

A different philosophy:

- Oriented to transactions rather than ‘bus cycles’
- Uses (semi-) independent channels to send information
  - Each channel is unidirectional
  - may be pipelined
- Latency may be many cycles
- Throughput improved by data bursts
- May have out-of-order transaction completion.
- Multi-master: in fact closer to a network than the traditional bus.

- A write transaction comprises a write command (address, burst size) accompanied by a burst of write data and concludes with a response which may signal an abort.
- A read transaction is similar but the data burst and status response are returned together.
- A transaction ID on each channel allows elements from multiple outstanding transactions to be matched appropriately.

AXI

AXI is more like a network than a bus. Transactions can be initiated from various units and will arrive at various destinations. In between they may be arbitrated and multiplexed as desired. The packet IDs allow steering so that the correct response is returned to the correct initiator.

Example: Read transaction

- Master sends a command on read channel specifying an address, data size and burst length. Command also has an ID tag.
- … other things may happen …
- Returned data burst arrives with appropriate ID tag and response status.
  - If okay, routed appropriately.
  - If abort recovery may be complex, including receiving but discarding later data packets already in transit.

AXI: pipeline detail

Data can be pipelined to reduce the distance travelled per clock cycle and, consequently, allow faster clocking and higher throughput.

Protocol

- Data in a stage asserts valid, downstream.
- A stage which will accept data asserts ready, upstream.
- If valid and ready are both active, a transfer takes place.

This is faster than, for example, a handshake which might go through several states and take (no fewer than) four clock cycles to complete one operation.

Data can move on every cycle if a pipeline stage can accept and pass on data simultaneously. They may work on this assumption, providing they can cope with buffering data even if the output is denied.
AXI-like pipeline

Consider a synchronous AXI pipeline stage.
The intention is to pass data on every clock cycle.
Data moves across an interface if both valid and ready are active.
If you indicate (upstream) you are willing to accept data (ready) that is a commitment.

There is not time to propagate a control signal throughout the pipe!

- Solution 1
  - Don’t indicate possible acceptance until you are empty
  - Benefit: simple to design
  - Consequence: the pipeline will never be more than half full

- Solution 2
  - Be prepared to accept new data even if you couldn’t pass on the current packet
  - Benefit: full bandwidth available
  - Consequence: twice as many flip-flops in each stage, (half are normally unused)

Single buffer per stage

If a blockage propagates backwards at one stage per clock data in adjacent latches will collide – some data will be lost.

With sparser occupancy data can stop safely; however throughput is reduced.

Two buffers per stage

With extra buffering it is possible to achieve ‘full’ throughput and still stall the pipe locally.

The disadvantage is the overhead in extra latches.

Note that in some pipelines there will be buffering implicit in the architecture to ‘even out’ such flow irregularities. Examples could include network routers storing and forwarding packets.

This is much like the traffic on a road.
**Simple example:**

This is the ARM device used in the laboratories. It uses:

- AHB interfaces for the high-performance devices
- a bus switch to facilitate parallel operations
- APB for the low-performance peripherals

---

**Example SoC**

The example in the slide is the Atmel AT91SAM9261 ARM-based microcontroller; this is the chip used in the laboratory equipment. The view shows the interconnection structure around the processor.

The processor masters two buses (instruction and data) which are fed into a bus switch matrix. Other devices can also be bus masters as the USB host interface, the LCD controller and the APB bridge all have DMA capability.

Dependent on the matrix are:

- APB
- ROM
- USB host and LCD controller (for programming)
- External bus interface
- RAM

The crossbar switch allows parallel operations so different masters can have access to different slave devices simultaneously. Clashes have to be resolved by inserting wait states.

**Bus occupancy can be reduced because the processor has:**

- separate instruction and data caches
- direct access to the on-chip RAM as Tightly Coupled Memory (TCM)

**Tightly coupled memory**

Tightly Coupled Memory (TCM) maps fast SRAM to specific addresses. This device has ten individually switched 16 KB blocks. This can allow parallel instruction and data access and still leave the I/O buses free for DMA.

TCM is sometimes preferred over cache in microcontroller applications because its timing behaviour is easy to predict. Cache accesses may be faster on average (as the hit rate may be better optimised) but predictability means that a worst case response can be guaranteed – important in some real-time applications.

**APB**

The APB hosts numerous lower performance peripherals. It may be run at a lower clock speed than the AHBs as a power saving measure.

**Bridge**

A bus bridge is simple a means of converting from one protocol to another. Usually a bridge is a slave on one bus and a master of another, although bidirectionality is possible.

**Split transactions**

When a bus structure becomes sufficiently complicated it can be an advantage to allow transactions to complete out-of-order. This gives decreased latency for some (urgent) operations at the expense of greater complexity, especially at the master where dependencies between reordered transactions may have to be resolved.

**Chip Multi-Processors (CMPS)**

Current generation CMPS typically share a bus to a level-2 cache. This is satisfactory for a small number of processor cores but as the number increases the pressure on this bus increases too. Such designs will not scale well. More elaborate – sometimes hierarchical – bus structures are evolving, although these exacerbate problems with maintaining cache coherency.

Another bus descends to the next level of memory hierarchy.
Network on Chip (NoC)

With integration levels increasing, simple bus structures become inadequate. Starting to develop networks on chip.

There broadly fall into two categories:

- **2D grids**
  - conveniently make regular structures on silicon surface

- **‘random’ networks**
  - like ‘conventional’ computer networks
  - may be packet- or circuit-switched

GALS

As clock speeds increase and wiring delays become more significant it is difficult to maintain a synchronous clock model across a whole chip. This problem was discussed in the section on timing (q.v.).

However one solution to this problem is to allow different IP blocks to be clocked independently with an arbitrary phase and, possibly, at different frequencies. It is then the job of the interconnection to cross the clock domains.

This form of interconnection is known as GALS (Globally Asynchronous, Locally Synchronous). GALS frees the SoC designers from a number of timing constraints which makes timing closure much easier. Each block is developed as a synchronous circuit but there is no need for chip-wide skew-free clock distribution.

Another advantage is the ability to run each block at its own ‘best’ frequency with the possibility of consequent power reduction.

There can also be a reduction in power supply noise. In a synchronous circuit logic begins to switch just after each active clock edge. Typically the number of gates switching over time diminishes during the clock period because not all logic paths are the same length. When gates switch they pull charge from the power supply or dump it onto the ground. The demand for charge (a.k.a. “current”) therefore varies periodically setting up a regular AC signal in the (extensive) power wiring. This both acts as a transmitting aerial (especially the wiring into the chip) and may affect other gates’ switching. If a whole chip is synchronous then this problem is at its worst; if there are several clocks with different phases (or frequencies) the demand tends to even out, reducing noise problems.

Handshaking

The simplest communication mechanism is synchronous on a one-item-per clock basis; this relies on assumptions that data will always be available and accepted on every cycle.

If data is not available on every cycle a ‘validity’ (or “request”) signal can be used to indicate when data is available.

If the receiver may not always accept data then some sort of flow control must be included. Across a synchronous interface – such as AXI, discussed earlier – this can be another status bit.

With an asynchronous interface various assumptions cannot be made and some form of handshake protocol is needed. This must be subject to synchronisation to the local clock, with a concomitant latency penalty.

Block transfers

A simple method of communication between asynchronous blocks is to synchronise each data request and, subsequently, latch the data from the bus. This results in a moderate latency but quite a low bandwidth because every transmission requires two synchronisations, one for the forward request and another for the reverse acknowledge.

Higher bandwidth can be achieved by buffering several data elements for a single synchronisation. The transmitter ‘owns’ a RAM into which it writes a message. When this is complete it passes the RAM to the receiver. After synchronising with the receiver’s clock the data can be read out at full speed. The overall latency is greater but the average bandwidth is also higher. This type of mechanism may be further enhanced (at additional hardware cost) by double buffering so that one RAM is filled whilst the previous one is emptied.

At its most extreme the interconnection may be asynchronous logic which can implement an elastic FIFO between transmitter and receiver. This could be a dual-port RAM which is written and read at different rates – synchronisation is only necessary when the FIFO is almost empty or almost full – or truly clock-free circuits.
Serial buses

This slide is something of an aside, in that it is chiefly concerned with systems off chip.

For wider system interconnection it is common to use serial interconnection:

- Inherently slower
- Far fewer chip-pins required
- Cheaper interconnection medium (wires, connectors, …)
- Suitable for wireless applications

Examples include:

- Ethernet
- USB
- I²C

On SoC

Pin restrictions do not apply to intra-chip connections.

Nevertheless the reduction in wiring is becoming attractive for some SoC applications.

Serial buses

In a serial bus transactions must occur as packets, so that the various signals are time-domain multiplexed onto the medium. Thus it may be that a transmitter sends a packet which contains ‘C’ bits of a command (such as read or write), ‘A’ bits of address (which may be a subsystem and/or a memory address) and ‘D’ bits of data.

**Ethernet**

Ethernet is probably familiar to you already. It is a peer-to-peer interconnection medium although the networks may be packet-switched.

**USB**

You are probably more familiar with USB (Universal Serial Bus) as a user than aware of its operation. It is a hierarchical structure where devices (slaves) are polled by the host (master) to allow them to transfer data. Data is communicated across a simplex (one direction at once) differential pair (see opposite) serial line.

Communication is asynchronous so each device has to have a precise clock reference matching the specification.

**I²C**

I²C (Inter-Integrated Circuit) is a Philips invention; to avoid legal complications it is typically referred to as Two Wire Interface (TWI) by other manufacturers.

I²C is a fairly slow interconnection, suitable for driving entirely in software with two PIO bits if required. It is typically used as a PCB level interconnection, for example for adding memory to small microcontrollers. However it is a multi-master bus where arbitration for mastery takes place via the same two wires.

Communication is synchronous as one wire is used as data, the other as a clock. However the ‘clock’ – really more a strobe – need not be regular as it may be software driven or paused by the receiving device if it is not ready.

Differential signalling

A differential signal is where a single logic state is represented by two digital wires which are always in ‘opposite’ states. Legal states are low/high and high/low.

The state of the signal is interpreted by looking at the difference between the wires, which will either be positive or negative – a binary choice.

Differential signalling is used for noise immunity. If two wires are physically close to each other any induced noise is likely to affect them in a similar way. A single wire compared to an unmoving ‘ground’ signal may have its state altered but the difference should be (largely) preserved. This is known as common-mode rejection.