Asynchronous Macrocell Interconnect Using MARBLE

W.J.Bainbridge * and S.B.Furber
Department of Computer Science
The University of Manchester
Oxford Road, Manchester, M13 9PL, UK
{bainbriw,sfurber}@cs.man.ac.uk

Abstract

This paper introduces MARBLE, the Manchester AsynchRonous Bus for Low Energy, a two channel micropipeline bus with centralized arbitration and address decoding which provides for the interconnection of asynchronous VLSI macrocells. In addition to basic bus functionality, MARBLE supports bus-bridging and test access, demonstrating that all the functions of a high speed macrocell bus can be implemented efficiently in a fully asynchronous design style.

MARBLE is used in the AMULET3i microprocessor to connect the CPU core and DMA controller to RAM, ROM and peripherals. It exploits pipelining of the arbitration, address and data cycles, together with spatial locality optimizations and in-order split transfers, to supply the bandwidth requirements of such a system. The design of a MARBLE initiator data interface used in the AMULET3i is presented, including a Petri-net specification suitable for synthesis using the Petrify tool.

1 Introduction

Asynchronous VLSI design has now reached the point where it is feasible to design complex ASICs and macrocells such as microprocessor cores in addition to simpler peripherals. There is therefore a growing need to find a regular way to interconnect asynchronous macrocells and to interface them to other (possibly synchronous, possibly off-chip) subsystems in the manner illustrated in Figure 1. The problems involved in the design of such interconnect are typical bus-design problems, although the nature of these problems is different from those encountered in previous bus designs due to the asynchronous VLSI aspect and the requirement for an order of magnitude improvement in the performance compared to typical backplane buses.

This paper discusses a system bus design based on the micropipeline design style [1] that is suitable for implementation in CMOS. Section 2 gives an overview of the split-transfer style used by the Manchester AsynchRonous Bus for Low Energy (MARBLE) and then the key issues of arbitration, signalling and data-validity are addressed in sections 3 and 4. Figure 1 shows a typical target environment for MARBLE, illustrating how support can be provided for Harvard architecture cores; the local-bus RAM is only necessary in high performance systems for which a general system bus such as MARBLE cannot satisfy the bandwidth and/or latency requirements. In addition to basic initiator-to-target interconnection, modern buses are also required to support advanced system level features such as hardware aborts and low latency bridging to other on-chip buses and the outside world. Section 6 of this paper provides details of how these are handled in MARBLE. Section 7 presents the design of a MARBLE initiator data interface as used in the AMULET3i chip, and sections 8 and 9 introduce extensions to the MARBLE specification which are currently under consideration, and a review of the design tools used in the specification and implementation of the interfaces used in AMULET3i.

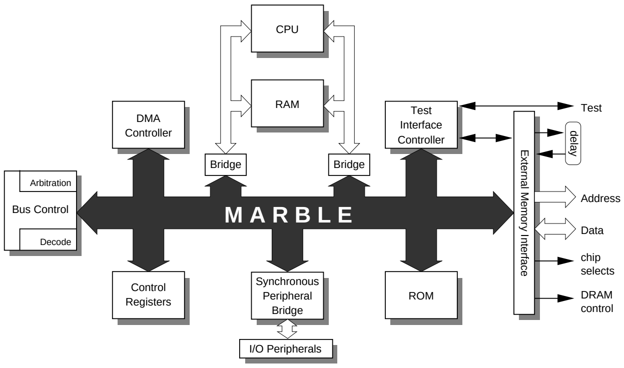


Figure 1: MARBLE System

2 The MARBLE Architecture

The benefits of pipelining in microprocessor design are well known [2], and similar benefits have been ob-

* Funded through a studentship from the EPSRC

served in asynchronous processors [3] and synchronous buses[4, 5]. Asynchronous buses present the same opportunity to pipeline the arbitration, address and data cycles of consecutive transfers. However, the lack of synchronization between these cycles introduces problems into the control of bus handover between initiators.

To illustrate how the address and data transfers may be pipelined, consider the bus interface arrangement shown in Figure 2(a). When an address is transferred, the address channel is occupied until the target device accepts the address and completes the handshake. A similar situation exists for the data channel. This tightly couples the bus protocol to the device signalling protocol.

The introduction of a latch into the bus interface at each port where a packet can be taken off the bus, as shown in Figure 2(b), decouples the bus from the device. This allows address packet *n* to be held in the latch at the target, freeing the address channel to allow the initiator to send packet *n+1*. At the same time data packet *n* can be transferred. Pipelining of the address and data cycles of different transfers is thus allowed but not enforced.

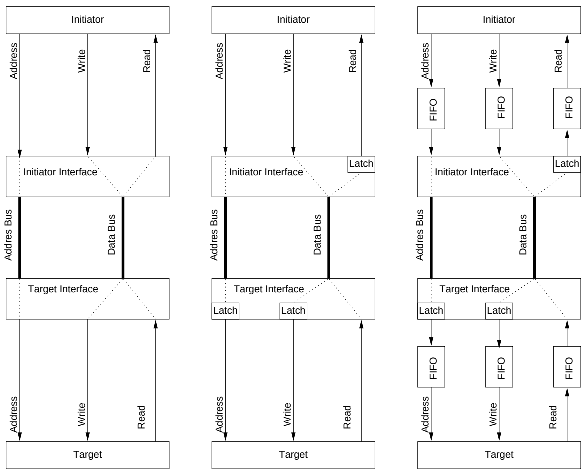
The arbitration for the address and data channels can be pipelined to occur with transfers on the channels in a similar manner, with the arbitration grant interpreted as *'you may use the channel when it is next idle'*. This allows the arbitration for the next access to be hidden behind the current cycle.

However, since the bus has multiple initiators and targets, and each of these could have any number of FIFO stages as shown in Figure 2(c), the decoupling between address and data cycles provided by the latches can potentially allow the following complex behaviours:

- Data transfers may complete in a different order from the corresponding address transfers;
- Multiple addresses could be streamed from an initiator FIFO to a target FIFO during a burst, then there could be a delay before the data transfers occur.

Simple in-order buses generally cannot cope with these behaviours and therefore include some form of *throttle* to regulate the address-data interlocking. Those in-order buses that allow these split-transfers can usually only do so by adding an extension to the underlying bus and passing other signals around the side of the bus, as opposed to having inherent support. Such a technique can be used to implement split transfers on AMBA [4] by using the *retract* command. The Split Transfer Asynchronous Macrocell INterconnection Architecture (STAMINA) [6] takes an alternative approach, using a split-transfer as its only primitive transfer mode. The principal characteristics of STAMINA are:

- it has separate address and data channels;



(a) (b) (c)

Figure 2: Pipeline Interface Scenarios

- it exploits the low latency and cost of asynchronous VLSI arbitration using mutual exclusion elements (MUTEXs) to provide the separate arbitration of the address and data channels;
- it only allows initiators to arbitrate for, and start cycles on, the address channel;
- it only allows targets to arbitrate for, and start cycles on, the data channel (thus write data is *pulled* and read data is *pushed*);
- initiators start address cycles as soon as address information is available;
- targets start data cycles in response to address cycles, immediately for writes, or when the data is ready for reads;
- a tag is passed with each address/data packet, indicating the initiator of the address cycle and the destination of the corresponding data cycle;
- a colour (sequence number) is passed with each address/data packet to enable data to be reordered to ensure that it is presented to initiator/target devices in the correct order.

Arbitration for data cycles only at the target reduces the arbitration requirements when compared to a more usual split transfer system, which would require all interfaces to be able to start (and hence arbitrate for) data cycles. It

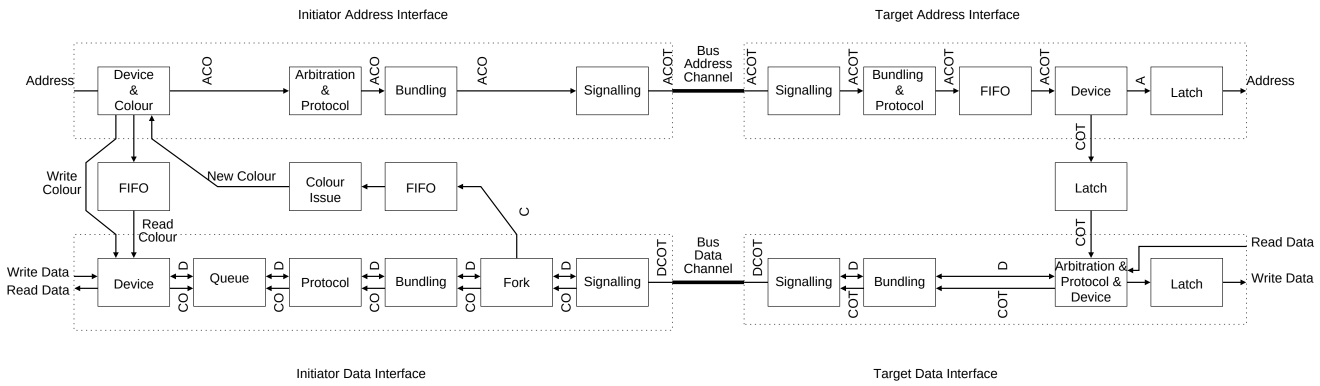


Figure 3: STAMINA Architecture

also means that there is no requirement for queues/reorder-buffers at the target interfaces since they will always perform data cycles in the required order, thus only the initiators need any reorder capability. This does, however, permit the situation to occur where the bus can be stalled waiting for data if the initiator commences a write operation but does not yet have the data available. This situation should be rare in practice, and once the data becomes available the system will continue its operation. Figure 3 shows how the initiator and target interfaces of a STAMINA bus can be decomposed and it also shows the locations of the latches, FIFOs and queues (reorder buffers) in such interfaces.

The FIFO in the target interface, as shown in Figure 3, is used to control how many transactions can be issued to the target before it will stall the address channel. The queue in the initiator interface is used to reorder the data packets so that data leaves/arrives at the initiator device in the correct order, but can be transferred across the data channel of the bus in any order.

Each packet on both the address and data channels carries with it three pieces of system overhead: an *initiator tag* (T), a *colour* (C) and a *operation* bit (O), all of which must be stored at the target so that the same tag, colour and operation (data-direction) are used for the corresponding address and data cycles.

The tag is used to identify which initiator the transfer originated from and is fixed to a unique value for each initiator. This is used to route the data cycles to the correct initiator (the one whose tag is on the data packet).

The colour is used to reorder data packets at the initiator data interface. The number of different colours for an initiator is determined by the number of outstanding addresses that it can issue before it requires a data cycle.

The operation bit is used to indicate whether the transfer is a read or a write, thus giving the direction of the data transfer. This information is required by the target peripheral and the control modules in the data interfaces at both the initiator and the target.

MARBLE implements a subset of the STAMINA [6] with only one colour (hence no signals are necessary to convey the colour), with the consequence that after an initiator has performed an address cycle it cannot start the next address cycle until a data cycle has been started to that same initiator, i.e. the initiator is never allowed to have more than one address issued that is waiting for its corresponding data cycle. This also removes the need for the queue and the FIFOs at the interfaces. Latches are used at the points where address/data is taken off the bus provide the necessary decoupling.

3 Multimaster Arbitration

Any multimaster bus requires a means by which an interface can obtain exclusive access to the bus. This is provided by the arbitration scheme, which may be distributed or centralized. In MARBLE two separate arbiters are required, one for the address channel and one for the data channel.

3.1 Distributed Arbitration

Distributed arbitration involves all contending initiators agreeing upon which one has been granted the bus. The SCSI bus standard [7] uses a distributed arbitration scheme where each device has a unique identifier. When a device requires bus access it must wait for an arbitration cycle and then assert its ID onto a set of signal lines. After waiting for a specified delay the device examines the lines to check if its ID is present or if that of another (higher priority) device has been placed on the lines. If the device's ID is present then it has won the arbitration, otherwise it has failed and should retry later. Futurebus [8] also uses a similar type of arbitration. An alternative form of distributed arbitration is that used in a token-passing network. Here a token is passed around all devices that may require access to the bus. A device may access the bus only when it is in possession of the token. Such a scheme requires complicated management of the token to ensure that one and only one token is present in the ring at any time, and in a low-power system the token should not be allowed to free-flow around the ring since this wastes power.

3.2 Centralized Arbitration

In a centralized arbitration scheme each device communicates with a central arbiter using a request-grant handshake. The device requests the bus, and the arbiter grants the bus to only one device at a time. This requires dedicated request and grant wires between each device and the arbiter.

Synchronous bus arbiters, as used in AMBA [4] and PI-Bus [5], sample the request lines on a clock edge and then apply a (prioritised) decision algorithm to determine which device is granted the bus. This approach cannot be used in an asynchronous VLSI bus since there is no clock, and it is prone to metastability.

MARBLE uses a centralized scheme with a bus-arbiter composed of two trees (one each for the address and data channels) of arbitrating call blocks as shown in Figures 4 and 5.

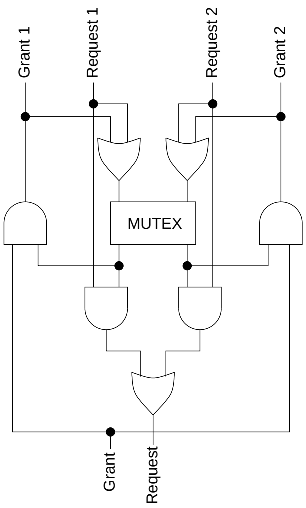


Figure 4: Arbitrated Call Block

The arbitrating call block provides mutually exclusive access to the bus by ensuring that the mutex is not released until the grant has fallen. The circuit shown in Figure 4 is a suitable implementation which guarantees that the mutex is not released until Request and Grant are both low. This is necessary to ensure that every arbitration propagates to the root of the tree, thus preventing any one initiator (the processor address interface for example) from hogging the bus.

If the root node request is immediately fed back to the acknowledge then the root node of the tree can be reduced to a simple mutex. However, the tree in Figure 5 is more general and allows the root node request/grant pair to be connected to the bus controller or test unit allowing bus access to be regulated or single-stepped. Different tree structures, composed of two-way or three-way [9] arbiters, allow the bus-bandwidth to be apportioned as required.

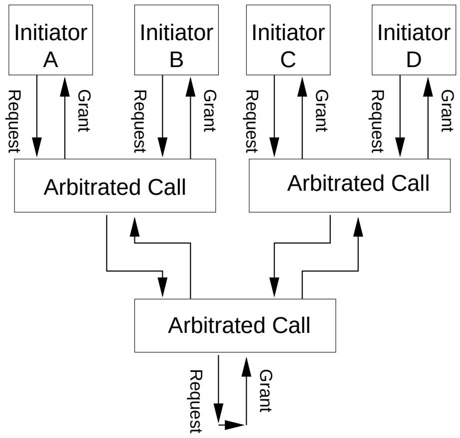


Figure 5: MARBLE Arbitration Tree

4 Signalling and Data-Validity

Although dual rail encoding provides fully delay insensitive operation it is considered too expensive for cost-sensitive applications, hence macrocells such as AMULET1 [3, 10] and AMULET2e [11, 12] use single rail signalling. The same approach to data communication can also be used for buses, although careful attention and simulation are required to ensure that bundling constraints are met. The issues that influence the choice of signalling protocol and data-validity scheme for a micropipelined bus include:

- The high loading of the bus lines requires large drive currents to maintain clean, fast edges. Since 2-phase signalling uses half the number of transitions of 4-phase signalling this is favourable from a power viewpoint, even though the control circuitry is more complex.

- Minimal transfer duration, so as to keep occupancy of the bus (a shared resource) to a minimum, is another area where 2-phase signalling may be favoured since

4-phase signalling has the overhead of the return to zero phases.

- Tristate data drive handover requires careful control and is closely linked to the clock edges in synchronous systems. Asynchronous systems must incorporate this handover into the signalling protocol which is easier to achieve using 4-phase signalling due to the additional edges which can be used (in conjunction with early data validity protocols [13]) to provide idle periods either side of the data valid region, thus avoiding drive clashes on the data lines.

- Tristate signalling drive handover is difficult when using 2-phase signalling since the new participant(s) in the transfer must be able to assume the current state of the bus signalling lines before commencing the transfer. When using a 4-phase signalling protocol this is not necessary since every transfer always returns the bus signalling lines to the same idle state and hence all devices enter every transfer in the same state.

- Wired-OR signalling as used in backplane buses [8] and early NMOS VLSI buses [14] cannot be used in CMOS technology when zero idle power is a design objective.

- Gated signalling using a wide OR gate to combine the individual requests from each interface to give the bus request line, and likewise for the acknowledge is possible when using 4-phase signalling, but the 2-phase equivalent results in a large, complex circuit.

- Tristate signalling would require charge-retention (negative resistance) on the signalling lines to maintain their value when the bus was idle or during an initiator or target handover period, but there is concern as to whether this would give sufficient noise immunity for the signalling lines, a spurious event on which may cause system failure.

Four phase signalling with narrow data validity is used in MARBLE. The initiator drives the address-request and data-acknowledge lines and the target drives the address-acknowledge and data-request lines. Information is bundled with both the request (push channel) and the acknowledge (pull channel) to allow bidirectional data transfer as illustrated in Figure 6, but information cannot be bundled during both phases of a transfer, thus only read or write data can be transmitted in a single cycle, not both:

- Address Push Phase - Address and other command signals are sent from the initiator to the target

- Address Pull Phase - 'Defer' (see section 6.2) status is sent from the target to the initiator

- Data Push Phase - Read data and the Abort Response is sent from the target to the initiator

- Data Pull Phase - Write data is sent from the initiator to the target

As shown in Figure 6, this data validity scheme provides a period of inactivity between successive uses of all data lines. This inactive period will be at least the time taken for a signal to propagate across the shortest interconnection provided by the bus, which with fast-off data drivers will avoid any drive-clashes on the bundled signal lines.

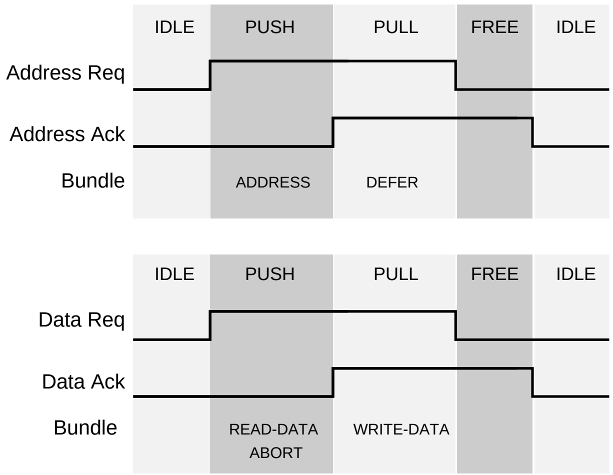


Figure 6: MARBLE Data Validity

Proposals to ensure that a system using tristate signalling operates reliably focus on ensuring that the signalling lines are driven at all times:

- Newly active interfaces begin to drive their signalling lines low immediately upon becoming active on the bus.

- Interfaces releasing the bus should use slow-off signalling drivers to provide overlap between their switch-off time and the next signalling-line driver's switch-on time.

- The centralised bus controller should use fast-on, slow-off pull-down drivers to clamp the signalling lines to ground when the signals are not driven by a device. The bus controller knows when lines are being driven since it is responsible for arbitration and decode.

However, although tristate signalling gives a more modular system (the tristates replace the large/distributed OR gate of the gated approach), the first incarnation of MARBLE for AMULET3i will use gated signalling and the tristate approach will be investigated in a later design.

5 Initiator ID Tagging

Every MARBLE transfer is tagged with the unique ID of its initiator. This is used to route the data cycles back to their originating initiator. The tag can also be used at the target for protection checks, for example, per-initiator access rights for the target can be applied. In addition, it facilitates the optimisations discussed below and can be of use in system testing.

6 Advanced Features and Optimizations

6.1 Spatial Locality

Some target devices and memory systems, notably DRAM, can operate with reduced cycle times for all but the first access in a series of accesses to the same or sequential addresses. In DRAM, this is known as paged mode access, where the first access requires both a RAS and CAS precharge, but subsequent accesses to the same page require only a CAS precharge which is typically $\frac{1}{3}$ or $\frac{1}{2}$ of the total RAS-CAS access time. To exploit this behaviour in a macrocell-based system requires additional bus support to convey the spatial locality properties of the addresses. It is preferable to communicate this information in advance of the sequential access when possible, since detection of sequential addresses when they arrive at the target is both expensive and too late to be of much use. MARBLE uses a two-bit code transmitted as part of the address packet to indicate the three situations where these types of optimization can be beneficial, as shown in Table 1.

It should be noted that although Table 1 shows both predictive and non-predictive spatial locality codes, the predictive ones may be used only when it is known that the transfer to the predicted address will be the next transfer on the bus (i.e. the bus is locked by the initiator), the final cycle of a burst having a non-predictive code. This is satisfactory for the important cases of cache-line fetches and DMA transfers. However, the primary use of the bus is typically for the instruction fetch by the microprocessor core. These transfers will be conveyed across the bus in single cycle arbitrated transfers, and the sequential flag should be used as appropriate, the target being responsible for filtering the code based upon the transfer's initiator ID tag (transfers that have a tag different to the previous transfer must be treated as non-sequential).

Table 1: SEQ and PRED Spatial Locality Coding

PRED	SEQ	Meaning	Typical Use
0	0	Non sequential	Random access
0	1	Sequential to previous address	Instruction Fetch
1	0	Next address will be in same 2^n page	Cache Line Fetch wrap-around
1	1	Next address will be sequential and in same 2^n page	DMA transfer, Cache Line Fetch, Block Move

6.2 Defer

The protocol of a multimaster bus should ideally include support for a 'hardware retry', where the transfer is not completed but is retried automatically by the initiator some period later. Its inclusion provides for more advanced features (that may not be required in many systems):

- Defer is the basic primitive necessary for the support of split transfers by simpler buses, which also require sequence tagging information which can be conveyed externally to the bus.
- It allows the bus to be released after starting a transfer to a device that takes a long time to complete, so other initiators can use the bus whilst deferring the completion of this transfer. This provides reduced latency to these devices and improves overall system performance.
- When a device is busy (possibly in a recovery cycle) and cannot accept another transaction, in a system without defer any attempt to access this device across the bus would result in the bus being stalled with the attempted transfer sat on it. Defer allows the initiator to be told to retry the command later, i.e. defer starting the transfer.
- Bridges between multimaster buses require that one or both of the buses connected to the bridge supports deferral. To illustrate the necessity of this feature, consider the case where device A on bus 0 tries to communicate with device C on bus 1 at the same time as device D on bus 1 tries to communicate with device B on bus 0. Both transfers require the use of the bridge and the two buses, so the bridge must reply with a defer command to one initiator to allow the other transfer to complete, otherwise the system would simply deadlock.

MARBLE supports a defer response through the DEF bit bundled in the pull part of the address cycle. It is only intended to support the last use listed above, since the others are either already incorporated into MARBLE's underlying split transfer protocol and decoupling latches, or are considered unnecessary in MARBLE's environment. Use of defer does incur some degree of power-burn due to the retries, but provided that the arbitration tree is fair (as is the arbitrated-call tree presented earlier), the worst case number of retries can be predicted since the only reason for deferring is that another initiator wants to use the bus, and so once that initiator has completed its activity the deferred initiator will eventually be serviced.

6.3 Abort

Many of the advanced features provided by a Memory Management Unit (MMU) are not necessary in an embedded system. Such features include virtual-memory support. However, there is often still a requirement to trap non-mapped memory addresses or memory protection violations using a simple bus-protection mechanism and both systems require a way to indicate external bus faults and memory failures.

MARBLE allows the signalling of errors as an abort response that is returned during the data cycle. This one bit signal, ABT, could be expanded to indicate the severity or location of the abort. One example of this use would be to indicate whether the abort was generated as a result of a memory access or a peripheral access. Since the abort can be used to indicate target device errors, it can also be used to support a software retry mechanism.

6.4 Concurrency Reduction

The composition of a typical MARBLE target interface is shown in Figure 7. This interface structure, through the inclusion of the three latches shown in the figure, provides decoupling between the bus address channel and the target device, between the bus data channel and the target device, and also between the bus address channel and the bus data channel. However, some situations can be envisaged where these latches are not required, and full coupling between the target device and the bus, and the two bus channels can be tolerated.

A good example of such a situation would be the system configuration registers. These are likely to be distributed throughout the chip, probably as many separate targets on the bus. Clearly since the target device here consists only of registers that can be read or written to across the bus, the latch in the data path is redundant. Removal of the other two latches (in the address interface, and between the address and data interfaces) causes the bus data channel activity for the cycle to be enclosed by the corresponding address cycle on the bus, thus tightly coupling the address and data cyles, forcing the removal of any pipelining of address/data cycles (but arbitration can still be hidden). However, since the target is an on-chip register that will be accessed infrequently, reducing the pipelining shouldn't impact overall system throughput too adversely.

The immediate hardware saving is obvious. Three

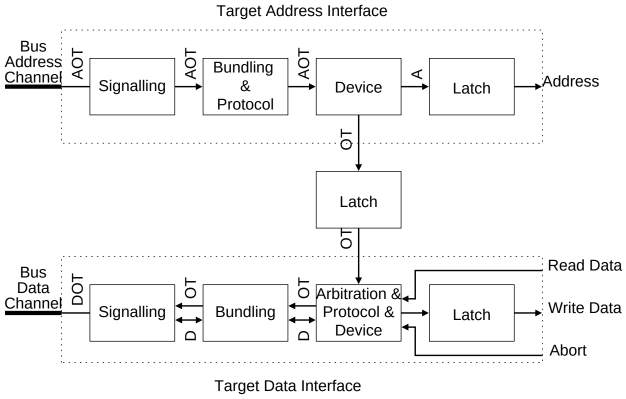


Figure 7: Target Interface Structure

latches (two of which have around 32 bits), and their associated latch controllers can be removed (although protocol converters to provide narrow-broad conversion may still be required). However, closer inspection reveals another beneficial hardware reduction: since the data channel activity is fully enclosed by the address cycle, only one of these *reduced targets* can ever attempt to use the bus at any one time, so the environment now provides the mutual exclusion between all of the *reduced targets*. This means that these targets can all share one input into the data-channel arbitration through a micropipeline call element [1]. Thus the additional hardware overhead for adding simple *reduced targets* such as control registers is very low.

7 MARBLE Interface Design

A set of generic MARBLE interfaces has been created to provide for the connection of devices using 4 phase broad protocol micropipeline interfaces. These are being used as part of the AMULET3i chip, but are designed for general use.

7.1 Interface Decomposition

A bus interface is a complex circuit that is required to connect a device to a bus ensuring that the bus protocol is obeyed. Since the MARBLE protocol includes pipelined address, data and arbitration cycles, the interfaces must include sufficient parallelism to accommodate the protocol, performing any conversion necessary if the device interface protocol is not compatible with the bus protocol.

The design of the MARBLE interfaces was partitioned into the functional modules shown in Figures 7 and 8. Each module was designed using Petri-nets to specify the behaviour, and then where possible these were synthesized using Petrify [15] to produce speed independent circuits. Some of the modules, notably the initiator-address-

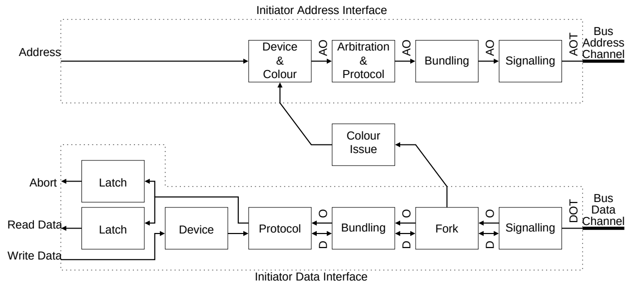


Figure 8: Initiator Interface Structure

signalling and target-data-signalling modules, were hand designed to ensure that any non speed independent behaviour was localized to these blocks and was carefully controlled.

Details of the design of the modules in the initiator data interface are presented below; (Figure 10 shows petri-nets for the modules, Table 2 lists the signals, and short descriptions of each block are provided). The design approach used for the other blocks was the same.

Table 2: Signals within the Initiator Data Interface

Signals	Meaning
SDR, SDA	bus request and acknowledge signals
SDS	bus data select signal from the central decoder
DRsig, DAsig	req and ack of channel between signalling and fork modules
DRcol, DAcol	req and ack of colour channel between fork and colour control unit
DRbund, DAbund	req and ack of channel between fork and bund modules
DRprot, DAprot	req and ack of channel between bundling and protocol modules
DonTR, DonTA	req and ack of bus data line tristate drivers
DoffR, DoffA	req and ack of data channel to the decoupling latch
RoffR, RoffA	req and ack of abort response channel to the decoupling latch
IDonR, IDonA	req and ack of pull data on channel
DonR, DonA	req and ack of data on channel from the target
O	operation signal bundled with the DRxxx and DAxxx signals and on the address and data channels of the bus, high=read, low=write
dummy0, dummy1	these are not signals, they are dummy transitions in the petri-nets

7.2 Signalling Module

The Signalling module waits until the data channel is active with a cycle destined for this initiator, as indicated by the Tag on the channel, and the SDS signal to the controller from the centralized address/tag decoder. The signalling module then allows this cycle to pass through to the next module (the Fork). The petri-net of the signalling module is shown in Figure 10(e).

The other role of the signalling handler is to isolate the non-isochronic fork nature of the bus data channel from the other control modules within the data interface.

If tristate signalling were used, the signalling module would be extended to include the control for the tristate and clamp enables for the Data Channel Acknowledge which it would be responsible for driving throughout the data cycle.

7.3 Fork

The Fork module in Figure 10(d) is a micropipeline fork, the implementation of which is a single C-gate. It forks off a connection to the initiator interface colour control unit which must be notified when the data interface starts a cycle so that it can throttle the address interface activity when necessary.

7.4 Bundling

The Bundling module, the petri-net for which is shown in Figure 10(c), ensures that the bundling constraints of the bus are met. In the initiator data interface this means that the bundling module is responsible for enabling the tristate drivers used to place write data onto the data channel at the appropriate time (during the pull phase). To isolate the problems of tristate drivers (which don't give an acknowledge to the enable line and are thus not delay insensitive) from the controller, the tristate driver is enabled by a Tristate Request, TR (DonTR in the petri-net), which is fed through a delay to give a Tristate Acknowledge, TA (DonTA in the petri-net), as shown in Figure 9. The length of this delay will have to be determined through simulation.

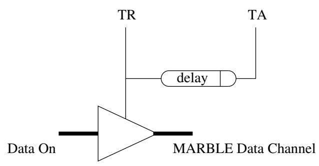
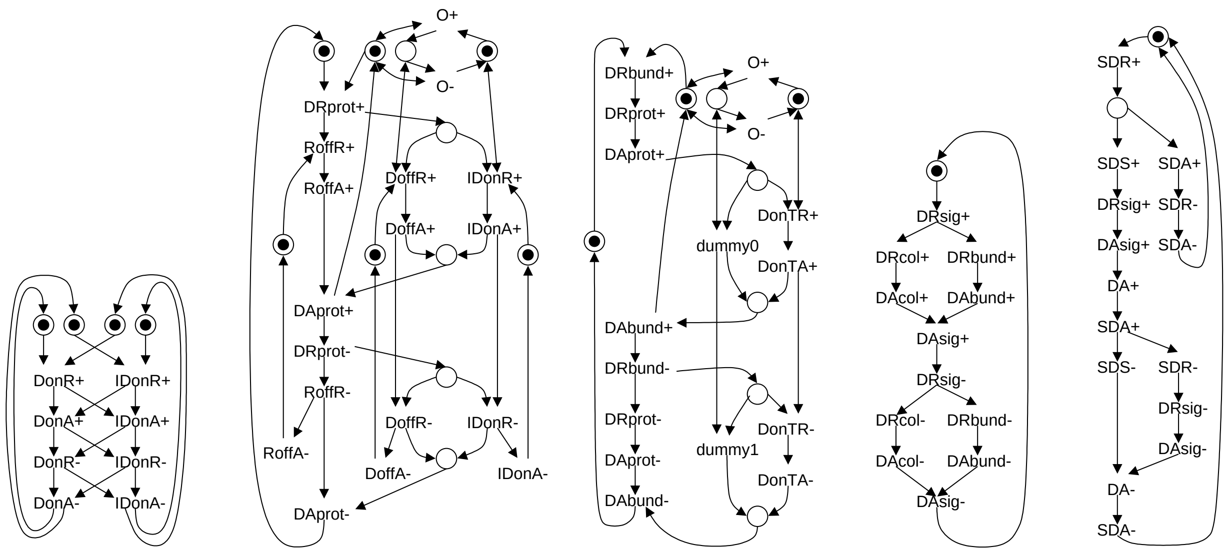


Figure 9: Tristate Driver



(a) Device (b) Protocol (c) Bundling (d) Fork (e) Signalling

Figure 10: Initiator Data Interface petri-nets

7.5 Protocol

The Protocol module in Figure 10(b) manages the signalling of the IDon (pull write data), Doff (push read data) and Roff (push abort response) channels, ensuring that data is provided to and removed from the bus at the correct time.

7.6 Device

The Device module is a protocol converter to connect the push channel supplying the write data from the initiator device to the pull write data channel from the Protocol module, which in practice amounts to a single C-gate. The petri-net for the device module is shown in Figure 10(a)

8 MARBLE Extensions

In addition to the support for the basic and deferred transfers described earlier, MARBLE will also support a number of functional extensions.

8.1 Test Strategy

MARBLE has been designed to aid VLSI testing through the inclusion of a test interface that allows the external memory bus to be used to enter test vectors. These can then be applied to individual macrocells across the bus and the results can be extracted by similar means. MARBLE supports two means of scan/test register access:

- Memory Mapped - the registers are mapped to an unused area of the memory map.
- Test Mode Transfer - an additional control signal is added to the address channel to indicate either Test Mode or Normal Operation.

The test interface is a finite state machine that acts as a MARBLE initiator that can:

- Read a vector address and then a vector from the external memory interface and apply these to a target.
- Read a vector address from the external memory interface and then fetch the contents of that address and write them to the external memory interface

It is also intended that the test interface will allow the external memory bus to be used to monitor the address-traffic from other macrocells in order to generate an address trace for software debugging.

8.2 External Memory Bridge (EMB)

The external memory bridge is a key feature of any embedded system whose memory requirements can't be satisfied using on-chip RAM. The EMB will be similar in nature to that used in AMULET2e [12], supporting direct connection to SRAM or DRAM and byte-packing to allow word fetches from byte-wide memory to allow bootstrapping from a single ROM (although this may also be achieved by using on-chip ROM). The EMB will use the SEQ and PRED spatial locality signals to control its DRAM RAS precharge, performing a RAS precharge only for non-sequential transfers. The issue of where the refresh circuitry should be located is still to be resolved; it can either be located within the EMB where it would require arbitrated access to the DRAM with any outstanding transfer, or it could be another dedicated initiator on the bus utilising MARBLE's arbitration.

8.3 Synchronous Peripheral Bridge

Although asynchronous design is becoming more common, much synchronous hardware is still designed or reused. There is thus a need to interconnect synchronous and asynchronous subsystems on the same chip. A bridge capable of interfacing MARBLE to synchronous macrocell buses like AMBA [4], and to older/slower legacy peripherals that use address-strobe type communication will be constructed as part of the MARBLE macrocell library. This bridge will be capable of being a bus-master on both of its client buses (MARBLE is one) and will use MARBLE's defer response as described in section 6.2.

9 Design Flow and Tools

The design methodology used for the MARBLE interfaces is a good example of the approach being adopted for the AMULET3 microprocessor, illustrating the present usability and limitations of asynchronous design tools. LARD [16] was used in the early stages for behavioural simulation, and to investigate the trade-offs between different protocols and modularisations. Then, having settled on a behavioural specification, the interfaces were specified using Petri nets. However, current synthesis tools are very slow with large Petri nets (due to state explosion) and lack hierarchy support, hence the design was manually broken into smaller modules as illustrated in Figures 8 and 7. Some of these blocks were hand-crafted, but many were then synthesized using Petrify [15].

However, synthesis tools automatically insert new signals to solve for a complete state coding (CSC). Inspection of the resulting design often shows that :

- The inserted signals are not where the designer would place them; often they are on the critical path of the circuit
- The output Petri net is so different from the one input by the designer that it is difficult to see any correlation between the two.

In many cases it was found that CSC could be achieved more effectively by changing the protocol of the controller outputs from the broad to the early protocol and slightly reducing parallelism within the design. This change then requires changes to the interface latch controllers since they must now perform the conversion from the early to the broad data validity scheme. Suitable long-hold latch controllers are documented elsewhere [17].

10 Conclusions

MARBLE, a macrocell bus for use in asynchronous embedded systems, has been presented. MARBLE is based on a split-transfer architecture allowing transfers between different initiators and targets to be interleaved without the need for retries, thus giving low energy operation and low latency. Novel support for spatial locality optimisations and test access are present in the design, as are the expected bus features such as atomic transaction and burst transfer support.

More advanced features including pipelining and bridging are supported but not rigidly enforced by MARBLE to improve system performance, and it has been shown how concurrency can be traded for hardware reductions in the design of systems using MARBLE. MARBLE will form a part of the AMULET3i chip, tapeout of which is expected during 1998.

MARBLE demonstrates that all the features of a high-speed on-chip macrocell bus can be implemented efficiently in a fully asynchronous design style. Asynchronous operation adds the advantages of elastic pipelines and zero quiescent power to the modularity and support for testability already offered by existing clocked macrocell buses.

References

[1] I.E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.

[2] J.L. Hennessy and D.A. Patterson. *Computer Architecture, A Quantitative Approach.* ISBN 1-55860-188-0. Morgan Kaufmann Publishers Inc., 1990.

[3] J.V. Woods, P. Day, S.B. Furber, J.D. Garside, N.C. Paver, and S. Temple. AMULET1: An Asynchronous ARM Microprocessor. *IEEE Transactions on Computers*, 46(4):385–398, April 1997.

[4] Advanced RISC Machines Ltd (ARM), UK. *AMBA, Advanced Microcontroller Bus Architecture Specification*, April 1997.

[5] Siemens AG, Germany. *Open Microprocessor Initiative (OMI) PI-BUS*, 1994.

[6] W.J. Bainbridge. *The Split Transfer Asynchronous Macrocell INterconnnection Architecture,*

STAMINA. Available from http://www.cs.man.ac.uk/-amulet/projects/STAMINA/index.html.

[7] American National Standards Institution. *Small Computer System Interface (SCSI)*, 1986.

[8] IEEE Computer Society Press. *FUTUREBUS : Specifications for Advanced Microcomputer Backplane Buses*, November 1983.

[9] D.J. Kinniment, A.V. Yakovlev, and B. Gaio. Metastable behaviour and system performance. In *Proceedings of the Second UK Asynchronous Forum*, June 1997.

[10] N.C. Paver. *The Design and Implementation of an Asynchronous Microprocessor*. PhD thesis, Department of Computer Science, University of Manchester, U.K., 1994. http://www.cs.man.ac.uk/amulet/-publications/thesis/paver94_phd.html.

[11] S.B. Furber, J.D. Garside, S. Temple, and J. Lui. Amulet2e: An asynchronous embedded controller. In *Proceedings of the 3rd International Symposium on Advanced Research in Asynchronous Circuits and Systems, ASYNC97, Eindhoven, Netherlands*, pages 290–299, 1997. http://www.cs.man.ac.uk/amulet/-publications/papers/async97_A2e.html.

[12] *AMULET2e Data Sheet*. http://www.cs.man.ac.uk/-amulet/AMULET2e_uP.html.

[13] A.M.G. Peeters. *Single-Rail Handshake Circuits*. PhD thesis, Eindhoven University of Technology, Netherlands, 1996.

[14] I.E. Sutherland, C.E. Molnar, R.F. Sproull, and J.C. Mudge. The trimosbus. In Charles L. Seitz, editor, *Proceedings of the First Caltech Conference on Very Large Scale Integration*, pages 395–427, 1979.

[15] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. PETRIFY: A tool for manipulating concurrent specifications and sythesis of asynchronous controllers. In *IEICE Transactions on Informations and Systems*, pages 315–325, 1997.

[16] P.B. Endecott. *Language For Asynchronous Research and Development, LARD*. Available from http://-www.cs.man.ac.uk/amulet/projects/amulet/lard/.

[17] J. Liu. *Arithmetic and Control Components for an Asynchronous System*. PhD thesis, Department of Computer Science, University of Manchester, U.K., 1997. Submitted for examination.