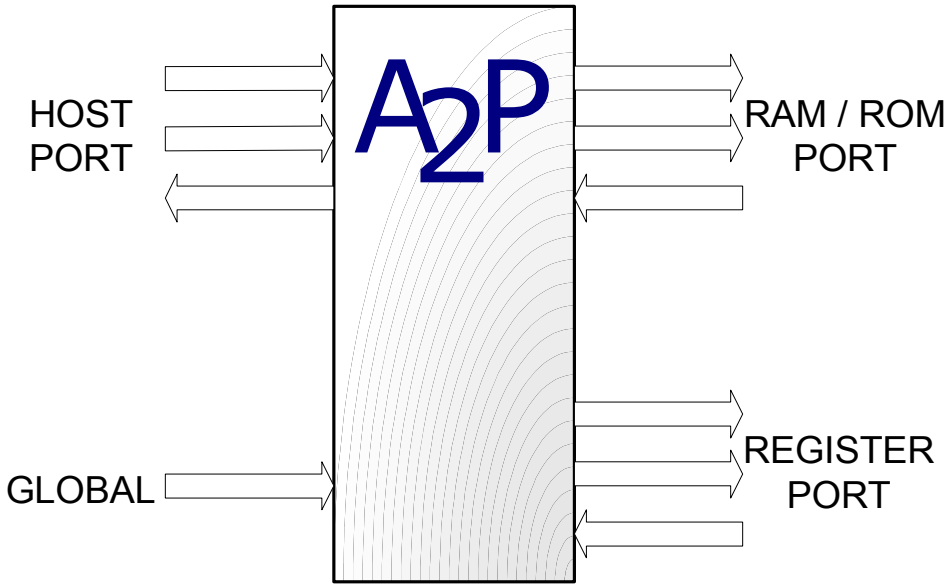


FEATURES

- High Performance 32 or 64-bit processor
- 16-bit compact instruction set
- Small size 0.065mm^2 at 130 nm (32-bit)
- Low power 0.02mW/MHz
- Extensible – custom instructions & addressing modes
- Supplied as RTL
- Multiprocessor Support
- Virtual Memory Support
- Full Tool Suite
  - ANSI-C Compiler, Assembler, Linker, Debugger
  - Instruction Set Simulator
- 4 Dedicated Accumulators (Extensible)
- 8 General Purpose Registers (Extensible)
- 512k I/O Registers (Extensible)
- Big and Little Endian Addressing Support
- Single Memory – Direct attach to RAM / ROM block(s)
- Upward compatible with all A2P families
- JTAG, 1(2) wire or “open” debug port

APPLICATIONS

- Replace older 8-bit and 16-bit controllers
- Set-top box
- Gaming
- TV,HDTV, DVR, DVP
- AV Player
- GPS
- Cellular
- Camera
- Imaging
- Embedded Control
- Tracking
- Protocol Offload
- Security
- Storage
- Network
- Wireless
- Smart Cards
- Multi-processor systems

OVERVIEW

A2P is a flexible processor architecture based on an innovative “fine-grain” microarchitecture configurability and an expandable instruction set. The flexibility of A2P enables a basic architecture to be developed for low power applications. Additional functionality, such as multi-CPU, DSP, floating point, SIMD and custom extensions are added to the base architecture in a simple plug-in style to expand functionality and exactly match each application’s specific needs. A2P also removes unused functionality to keep area and power optimized at all times. A2P is suitable for a wide range of applications, from ultra-low power, deeply embedded control functions to high performance multicore video processing.

A2P’s instruction set is optimized to execute complied C code directly and achieve the same performance as assembly level equivalents. SoC developers can now fine tune the processor hardware directly using C level algorithms as a reference, eliminating the typical steps of converting the C algorithms to assembler and then optimizing them around fixed hardware and inflexible instruction sets.

SYSTEM INTERFACES

A2P is agnostic to interconnect types and may support standards such as AMBA or OCP and Advanced Architectures' own A2R and A2B buses. There are 3 primary interfaces. The Host Port is for a host or debug port to be able to control the A2P for debug and system control. A JTAG port is provided in the base configuration. The RAM/ROM port supports the direct attachment of synchronous SRAM and SROMs to the A2P core. This interface may also be used to attach to a system bus to access other system memories if necessary. The Register Port supports the direct attachment of system registers to the core. This allows dedicated control of other system blocks that are in-turn controlled by these registers.

ADVANCED PROCESSOR AND SYSTEM TOOLS

The A2P Integrated Development Environment (IDE) provides for advanced A2P profiling and configuration as well as performing system level analysis. All the tools necessary for chip architects and designers to perform unified hardware/software co-development and co-verification are included:

ANSI C compiler, Assembler, Linker and debugger

Cycle Accurate Instruction Set Simulator

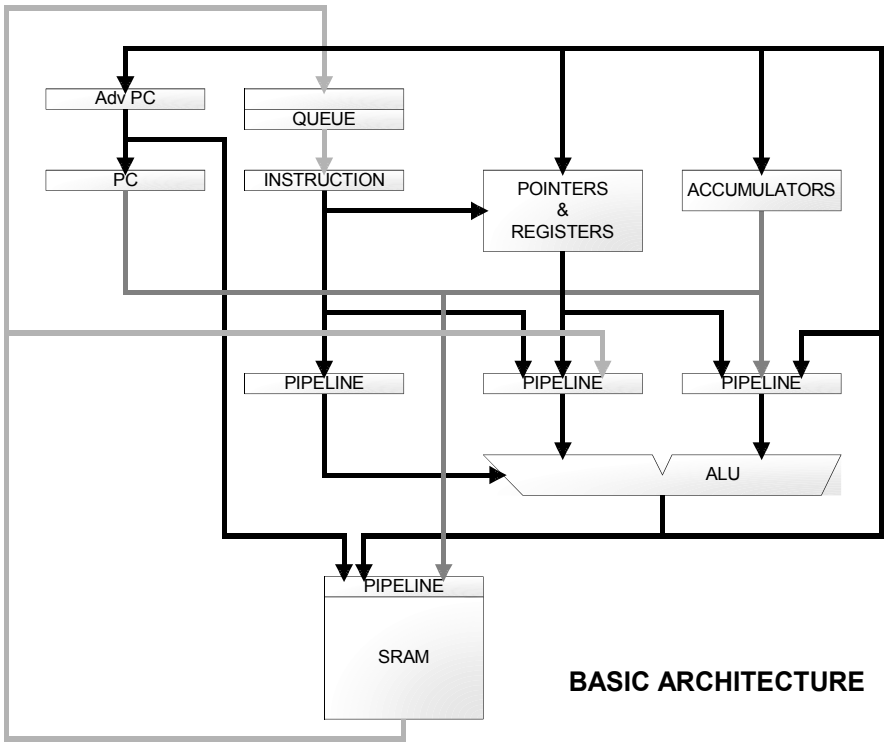
- attachment for peripheral descriptions in C/C++

True Multi-processor development, simulation and debug environment

The tools utilize a high speed Instruction Set Simulation environment. The hardware/software co-simulation enables the tool to collect key system performance measurements. These measurements are used to identify bottlenecks. The tool also performs software profiling as well as the collection of system performance indices. These measurements pinpoint inefficiencies in the design and enable the concurrent optimization of the system architecture, the hardware and the software. The tool models the complete memory hierarchy as well as the processor core. Modeling can be extended to other system modules by the attachment of C/C++ based models.

A2P ARCHITECTURE

A2P utilizes an accumulator and evaluation stack approach that is a merger of single address and stack architectures. Instructions may push and pop accumulators on and off of an evaluation stack. The evaluation stack rides in front of the procedure stack in memory but the A2P holds a portion of the top-of-stack inside the processor. This eliminates main memory references as operands are usually popped from the stack before being flushed to memory.



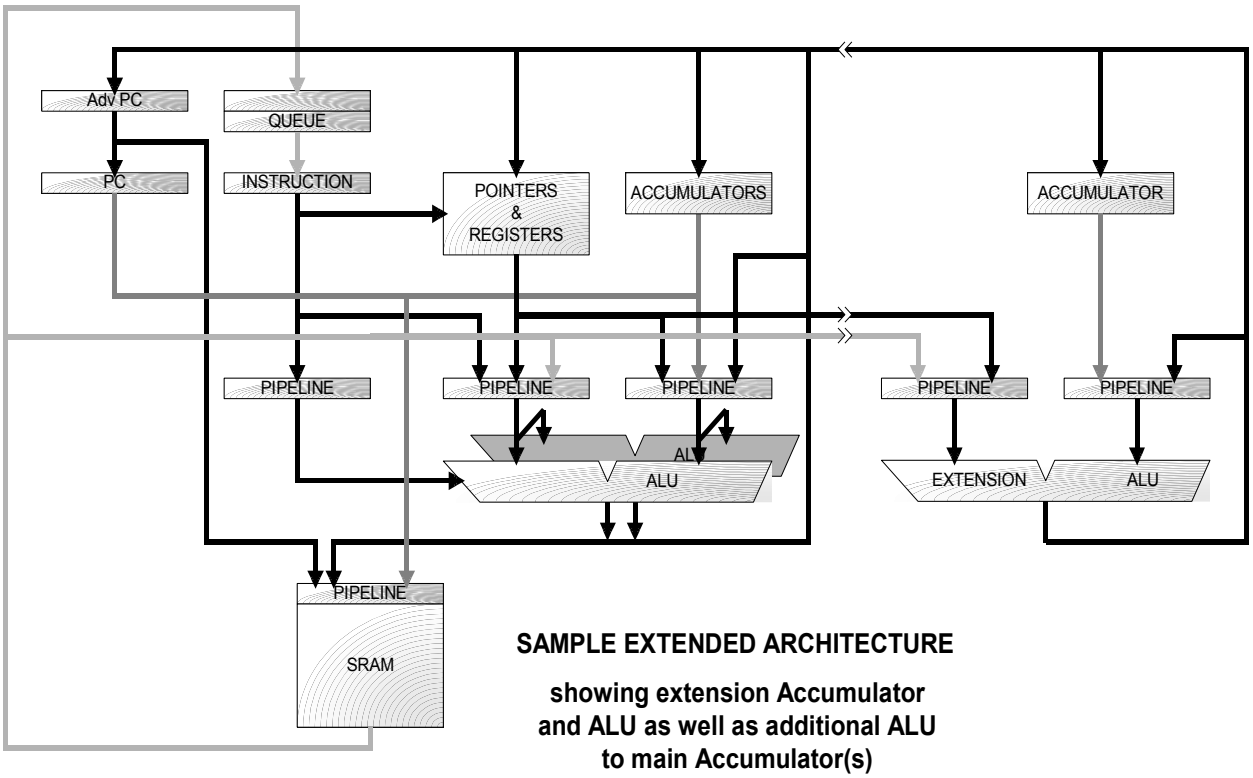
BASIC ARCHITECTURE

For basic designs, no register file is required, keeping A2P small and fast. Register files are added where high performance goals outweigh power and area considerations. A2P supports embedded memories for instruction and data, and/or caches to provide equivalent performance when larger and more remote memories are needed.

The instruction set is packed into 16-bit short-words, and prefixes can be used to extend the functionality for almost all base instructions. Basic A2P implementations typically need only a single memory for both code and data, further reducing the A2P footprint. A2P is fully capable of supporting separate code and data memories with or without caches.

Extensions to a basic A2P3 are achieved by adding an accumulator with its own set of functions. Examples of this are DSP MAC and floating point functions. A2P3 allows for 3 additional accumulators, one being reserved for floating point. Additional functionality is also achieved by introducing more functions to the basic ALUs and by specifying custom addressing modes. A2P3 supports complex (DSP-like) addressing modes included in the core so that code size can be kept very small. Also available is an SIMD engine that is capable of both fixed and

floating point arithmetic and may be configured up to 512-bits wide allowing a peak floating point throughput of 32 floating point operations per cycle and an 8-bit performance of 64 fixed point operations per cycle.



SAMPLE EXTENDED ARCHITECTURE

showing extension Accumulator and ALU as well as additional ALU to main Accumulator(s)

The figure above shows two simple extensions to the basic architecture. An additional ALU is added to the main ALU. This could be a single-cycle multiplier or a cryptographic S-box for example. Also shown is an Accumulator extension. This could be an FPU or a SIMD unit.

Many other more complex extensions are possible. Multiple Address Generation units can be added to make the processor look like a full DSP or a streaming floating point computation engine.

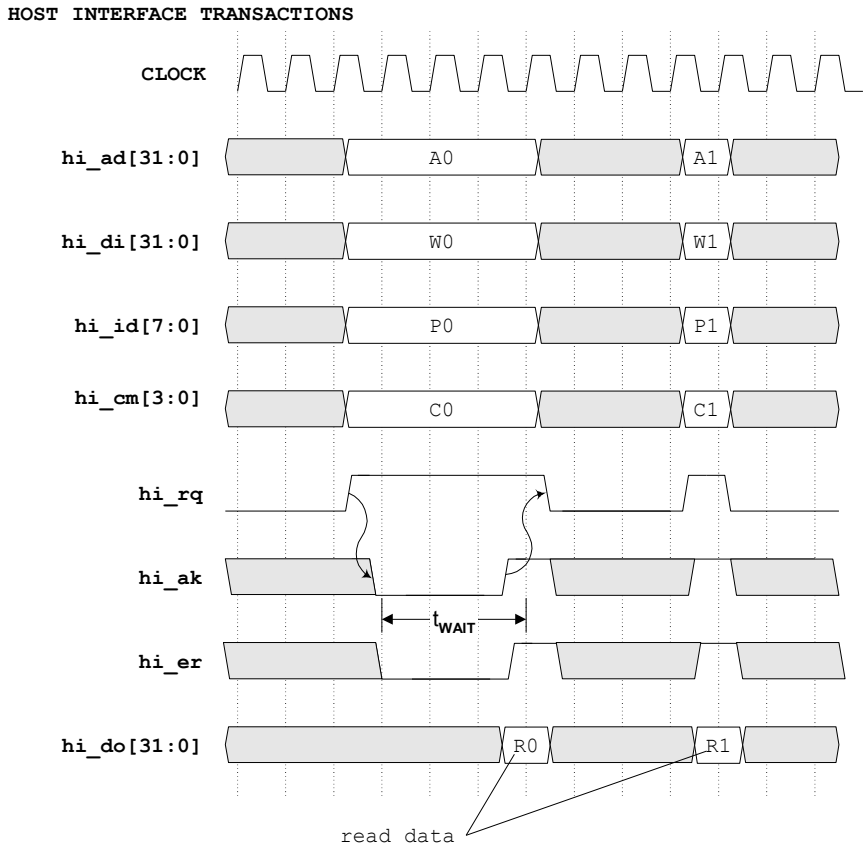
Typically to improve on basic system performance the SRAM can become either a combine code/data cache for a first level improvement and then separate code and data caches for best performance.

The basic premise of the extensibility of the architecture is to provide flexibility to the system architect so that the optimal trade-off between performance, power and silicon area can be readily achieved.

SIGNAL DESCRIPTIONS			
Signal Name	**Width**	**I/O**	**Description**
Host Interface			
hi_ad	32	In	Host Interface Address
hi_di	32	In	Host Interface Data In -- Data from the host to the A2P
hi_do	32	Out	Host Interface Data Out -- Data to the host from the A2P
hi_pid	32	In	Host Interface Processor ID
hi_cmd	8	In	Host Interface Command
hi_req	1	In	Host Interface Request
hi_ack	1	Out	Host interface Acknowledge
System Register Interface			
sr_ad	32	Out	System Register Address
sr_di	32	Out	System Register Data In -- Data to the Register from the A2P
sr_do	32	In	System Register Data Out -- Data from the Register to the A2P
sr_wr	1	Out	System Register Write
sr_rd	1	Out	System Register Read
sr_ak	1	In	System Register Acknowledge
sr_er	1	In	System Register Error
Data Memory Interface			
dm_ad	32	Out	Memory Address
dm_be	4	Out	Memory Byte Enables
dm_di	32	Out	Memory Data In -- Data to the Memory from the A2P
dm_do	32	In	Memory Data Out -- Data from the Memory to the A2P
dm_wr	1	Out	Memory Write
dm_rd	1	Out	Memory Read
dm_ti	3	Out	Memory Tag In -- Tag from the A2P to the Memory
dm_wk	3	In	Memory Tag Write Acknowledge
dm_rk	3	In	Memory Tag Read Acknowledge
dm_er	1	In	Memory Error
System Interface			
clock	1	in	System Clock
reset	1	in	System Reset
stall	1	in	Processor Stall. Maintains all flip-flops in current state
start	1	in	If high during reset initiates immediate processor start when reset finishes
wake	1	in	Wake processor up from sleep state
Interrupt	variable	in	External event signal(s)

HOST INTERFACE

The Host Interface provides debug, monitor and control of the A2P and its peripherals. All memories and registers within the system are accessible over this interface. The connection to this interface depends on the configuration of the overall system. In standalone A2P systems it is most often an interface directly to a JTAG port for off-chip debug and control. In systems that use an A2P as an adjunct to more general purpose processors, an interface to the control processor's bus mechanism is used. In multi-core A2P systems the Host Interface often connects to an A2R (Register Bus) that is used as system wide debug and control bus that in turn connects to a JTAG port. As part of the A2P release both a JTAG debug interface and an 8-bit processor interface are provided.



Accesses over the Host Interface are 16-bits, 32-bits or 64-bit data widths. The memories in an A2P system are always word oriented but their addresses are always byte addresses. For this reason the lower 1, 2 or 3 bits of the address are ignored. Inside the A2P instruction set there are instructions that manipulate bytes and half-words that use these 2 bits. A2P supports a register space of 2^{32} registers where each register matches the machine width of 32 or 64 bits. The Host interface addresses to the register space are to single register granularity.

Host interface signals include a Processor ID field that is used to uniquely identify one of up to 256 A2P processors in a system. This field may be extended to 16-bits if required. The Command field provides read and write

access commands to the various memory blocks in an A2P system. For a base A2P there are only two memory spaces, one for registers and one for memory. Optionally a separate Instruction Memory space may be provided to support true Harvard Architectures. Note that A2P's equipped with caches will provide direct access to the cache memories via the register interface. In systems that support virtual memory the host addresses are the physical addresses.

The Host Interface transactions are all synchronous to the A2P clock. Transactions are controlled primarily by the request and acknowledge signals "hi_rq" and "hi_ak" respectively. On the assertion of a request the interface will look for an acknowledge before the end of the current clock cycle. If the acknowledge is low then the access is extended until the clock edge that registers the acknowledge signal is asserted high. This allows for any length of transaction to be accommodated on the bus. Host Interface accesses to an A2P may be performed as cycle-steal operations, depending on the version of A2P, and as such their completion is dependent on the on-going processing within A2P. The second transaction shown in the diagram is a zero-wait transaction.Some A2P versions are capable of zero-wait state transactions. Note that when the acknowledge is asserted on a read transaction then the read data from the A2P is also asserted with the same timing. This simplifies the strobing of the read data into the transaction master interface.

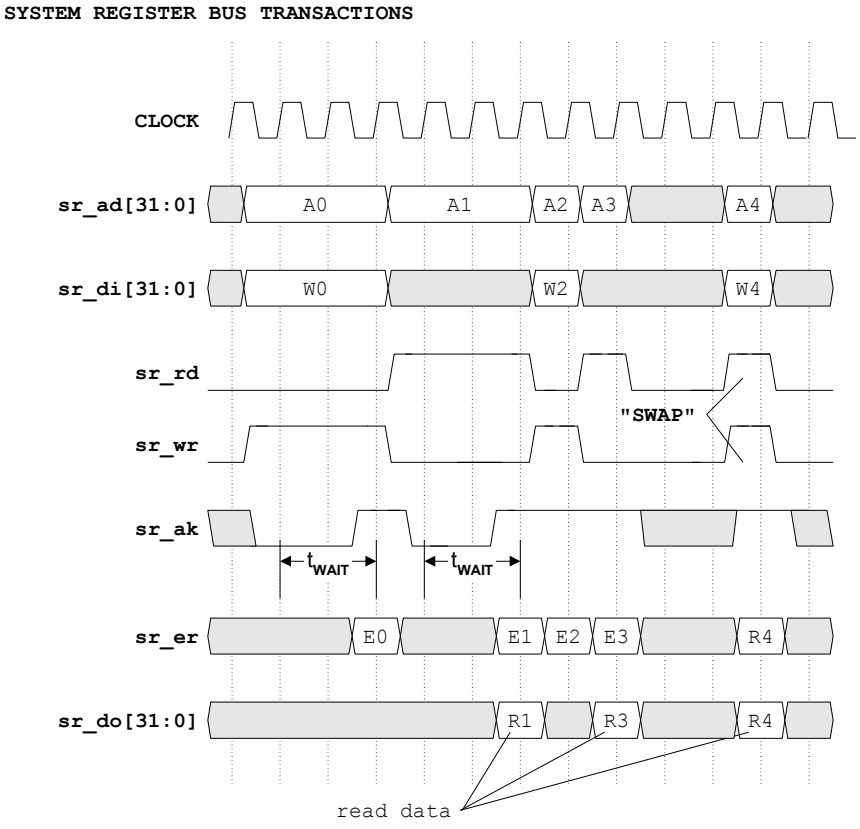
A2P may deny access to certain registers depending on the implementation. Normally, accesses are only denied if the processor is running. A halted processor will normally allow all accesses to all registers. When an access to a register is attempted to a running processor an the addressed register is not available the processor

will assert the error signal "hi_er" with the same timing as the acknowledge. This mechanism may also be used for passing on errors detected from memory accesses where the memory is protected by parity or other error detection mechanism.

The Host Interface is capable of supporting multiple hosts. For example, a JTAG based debug interface may coexist with and interface to an on-chip host processor. In this mode there is a simple priority scheme that grants access to one or the other host. In such a system, on the assertion of a request ("hi_req"), a grant signal is generated to the requestor based on whether or not the other host is active or not. If both hosts attempt an access at the same time then the losing host, the one that does not receive a grant, will back-off and remove its address, command, pid and write data fields until the grant signal is re-asserted. This is not shown in the timing diagram as this mechanism is not seen by the A2P.

SYSTEM REGISTER INTERFACE

The A2P has a flat register space of at least 2^{16} registers each with a width of the machine word size (16, 32 or 64). The first 64 registers are addressable from a single 16-bit instruction and are the designated as the “Primary Registers” of the A2P architecture. All registers above R63 are designated System Registers. Some of these registers are core registers of the A2P but the bulk are intended for processor extensions and closely coupled I/O modules. Access to these registers is performed over the System Register Interface or “sr_bus”.



The interface supports read, write and swap accesses to any and all registers that are on the sr_bus. Swap operations provide for an atomic exchange of data between a register and an A2P internal accumulator. Registers that are designated read-only or write-only must define their operation when a swap operation is performed.

The bus is single cycle but delays (wait-states) can be introduced by asserting the acknowledge “sr_ack” low in the first cycle of an access as signaled by “sr_wr” and/or “sr_rd”. If this happens the access will be extended until the acknowledge is asserted high and meets the set-up time before the next rising clock edge. In a simple system that only has single cycle accesses on the sr_bus the “sr_ak” line may be tied high.

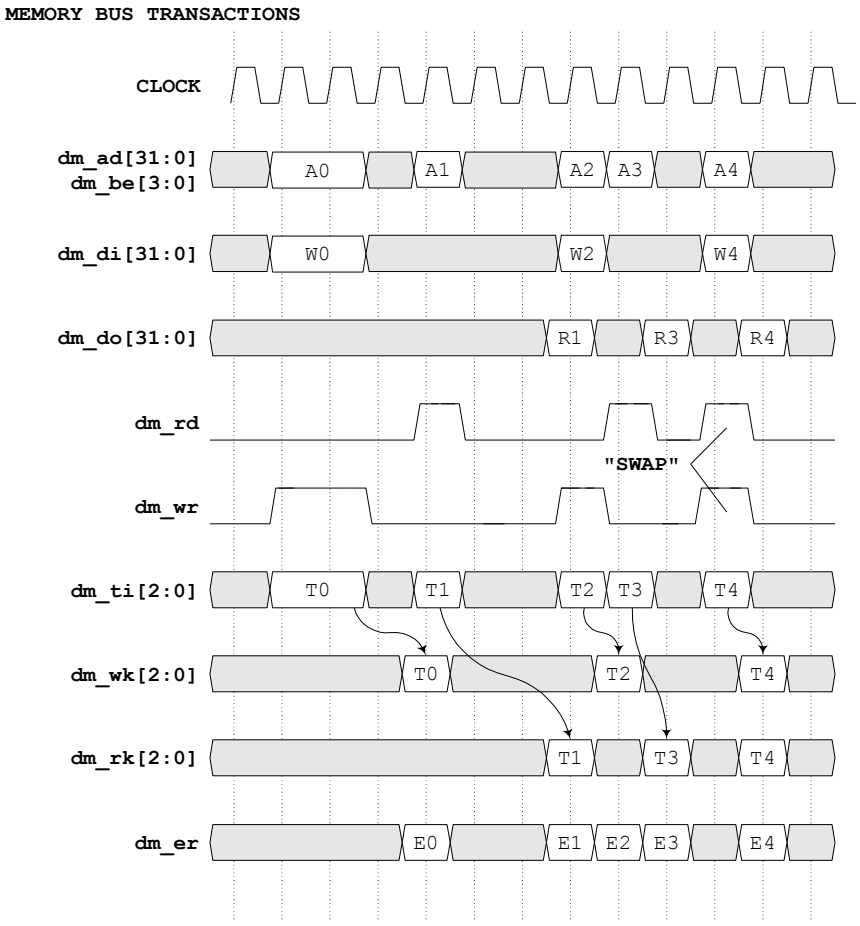
Note that the System Register Interface expects that writes happen synchronously with the rising edge of the clock following the assertion of “sr_wr”. Register reads are asynchronous. There should only be a combinatorial

multiplexor between the register and “sr_do” selected by the addresses on “sr_ad”. When a register or block of registers is deselected by the address or whenever “sr_rd” is low then zeros should be driven onto their individual “sr_do”. This allows for multiple modules to connect to the “sr_do” into the A2P by a simple 32-bit wide OR gate.

The “sr_er” signal provides the capability of flagging errors back to the A2P. It has the same timing as the read data on “sr_do” but is valid for writes as well as reads. It may safely be tied low if unused.

MEMORY INTERFACE

The Memory Interface may have between one and three instances on the A2P. In base mode there is a single combined code and data interface. More complex versions may have up to two data memory interfaces and a separate code interface. The interface protocol is a split transaction mechanism that allows for multiple outstanding accesses that may complete in any-order. Again in base mode multiple transactions are limited or not available but still conform to the protocol. Each access is accompanied by a tag value that identifies the unit within the processor that is performing the access. Accesses are terminated when read and write acknowledge tags are returned. The returning tag values are used to steer the incoming read data or write responses to the unit that initiated the access.

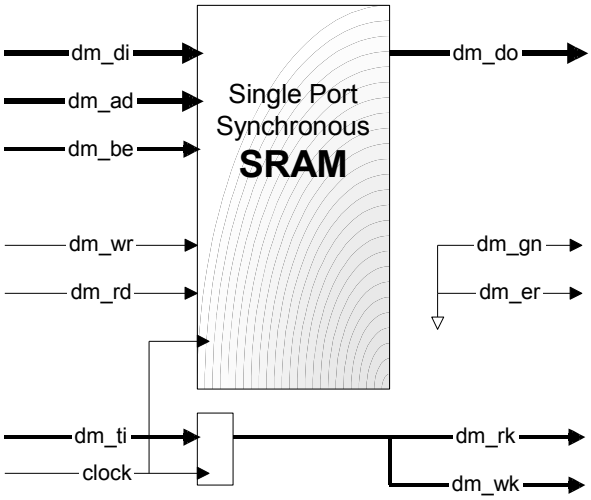


The Memory Interface issues signals ready to be clocked into a register. This is to allow for the direct attachment a synchronous SRAMs to the A2P core. When connecting this interface to a system bus then the interface signal output from the core can be simply registered before use as necessary. The “dm_rq” / “dm_gn” signals provide a means of either arbitration for the bus or direct attachment to a FIFO-style interface whereby the “dm_rq” functions as a FIFO push signal and the “dm_gn” functions as FIFO input ready (i.e. not full).

The swap operation is often NOT supported by synchronous RAMs and provision is made within the A2P core to issue a back-to-back pair of operations in order to synthesize this operation. The first access is issued as a swap but is immediately followed by a write. This requires that the SRAM operates such thatreads have priority over writes. The swap must cause the initiator to be given the right to hold the bus for a subsequent write access to ensure the atomicity of the access. For this to happen the arbitration logic that selects between the master devices on the interface must recognize that a swap has been granted the bus and grant the next access to the same master.

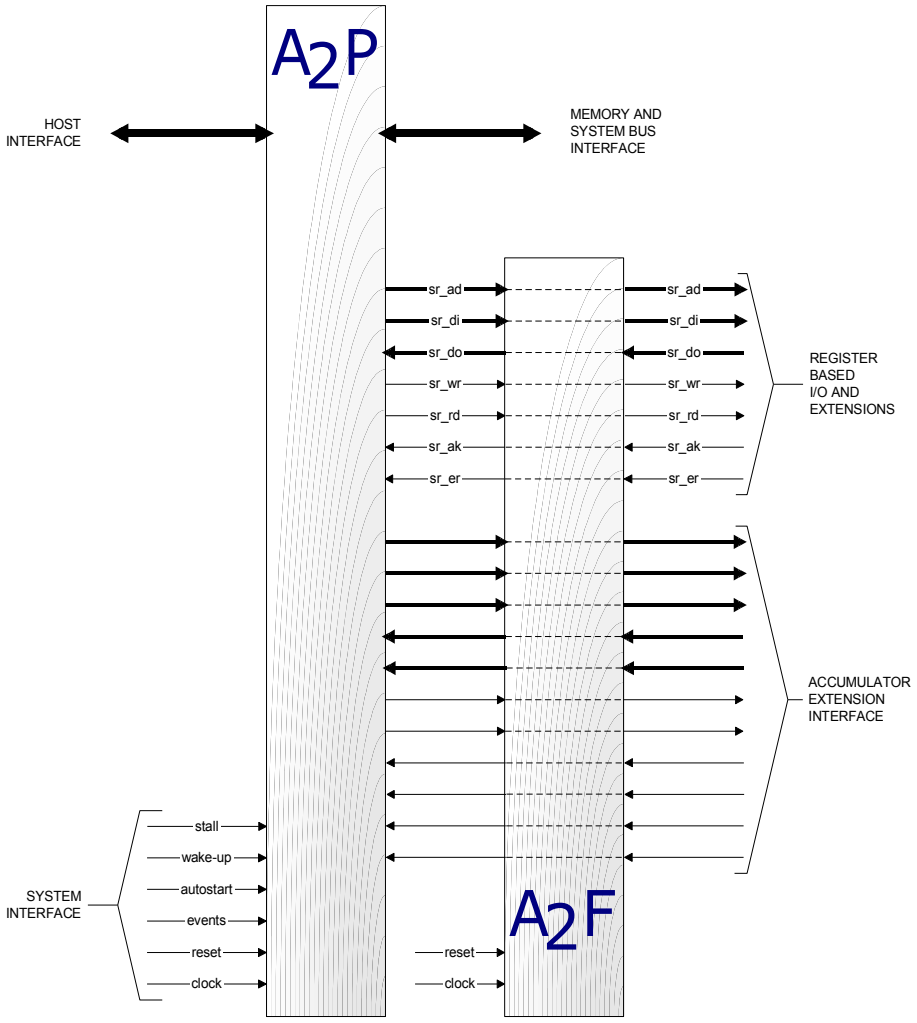
In many deeply embedded applications all that is needed is a single SRAM to serve code and data to the A2P. In these cases a Single Port Synchronous SRAM can be directly attached to the interface as shown below. Note that, in this style of application, if a multi-cycle access is required simply hold the “dm_gn” signal low until the RAM has completed its access.

DIRECT CONNECT TO SRAM



A2P EXTENSIONS

The A2P instruction set provides for a wide range of extensions. A detailed description of the extension interfaces is available in the “A2P Extension Interfaces” document.



Functions such as complex multiply-accumulates, network packet decollation and collation, and intricate addressing modes, may be added to a base A2P. Operations may also be added to the existing ALU functions given a chosen opcode to decode. More complex systems can include a new accumulator and a whole set of operators for that accumulator. Examples of these types of extensions are floating point and SIMD. Extensions may also have direct memory access assisted by a third type of extension; addressing. Custom address generators are added to the main data memory access channel or through a second memory access interface.

Addressing modes are added to the basic instruction set via reserved codes. Unique modes may be implemented by including a prefix for any other instruction. The diagram above shows the connection of the

standard floating point unit (A2FP) to the A2P. This unit attaches directly to the A2P Accumulator Extension Interface without any extra logic. The FPU includes control and status registers that detect and control floating point exceptions, and also control over rounding modes and exception masking. These registers are user visible and accessible through the System Register Bus. All inputs to the A2P that may be shared with other external units need only to be OR-ed together for input into the A2P. So the bus-like structures shown here are in fact OR trees in implementation.