

1. Instruction Set Simulator generic API

1. Structures and values

1. Enumerated values
2. Instruction request
3. Instruction response
4. Data request
5. Data response

2. Functions

1. void reset()
2. void getRequests(struct InstructionRequest &, struct DataRequest &)
3. uint32_t executeNCycles(uint32_t ncycle, const struct ?
4. bool virtualToPhys(addr_t &addr) const
5. void setWriteBerr()

2. Other APIs

1. Sideband signals

1. void setCacheInfo(const struct CacheInfo &info)
2. void setICacheInfo(size_t line_size, size_t assoc, size_t n_lines) ?
3. void setDCacheInfo(size_t line_size, size_t assoc, size_t n_lines) ?

2. Debugger API

1. unsigned int debugGetRegisterCount()
2. debug_register_t debugGetRegisterValue(unsigned int reg)
3. void debugSetRegisterValue(unsigned int reg, debug_register_t value)
4. size_t debugGetRegisterSize(unsigned int reg)
5. void dump()
6. void set_debug_mask() and the m_debug_mask variable

3. Implementation notes

1. executeNCycles semantics

Instruction Set Simulator generic API

This new API is an evolution of the ISS API. New features are:

- Use structs rather than arguments, this eases API evolution (which can be accomplished through carefully choosed default values);
- support for CPU modes;
- support for generic Memory management unit.

Structures and values

Enumerated values

Execution mode for any Instruction/Data access, checked by mode-enabled caches

```
enum ExecMode {  
    MODE_HYPER,  
    MODE_KERNEL,  
    MODE_USER,  
};
```

Operation type on Data cache access

```
enum DataOperationType {
```

```

        DATA_READ,
        DATA_WRITE,
        DATA_LL,
        DATA_SC,
        XTN_WRITE,
        XTN_READ,
};

```

When operation is XTN_READ or XTN_WRITE, address field must be one of these values, it determines the extended access type.

register name	index	description	mode
MMU_PTPR	0	Page Table Pointer Register	R/W
MMU_MODE	1	Data & Inst TLBs and caches Mode Register	R/W
MMU_ICACHE_FLUSH	2	Instruction Cache flush	W
MMU_DCACHE_FLUSH	3	Data Cache flush	W
MMU_ITLB_INVAL	4	Instruction TLB line invalidation	W
MMU_DTLB_INVAL	5	Data TLB line Invalidation	W
MMU_ICACHE_INVAL	6	Instruction Cache line invalidation	W
MMU_DCACHE_INVAL	7	Data Cache line invalidation	W
MMU_ICACHE_PREFETCH	8	Instruction Cache line prefetch	W
MMU_DCACHE_PREFETCH	9	Data Cache line prefetch	W
MMU_SYNC	10	Complete pending writes	W
MMU_IETR	11	Instruction Exception Type Register	R
MMU_DETR	12	Data Exception Type Register	R
MMU_IBVAR	13	Instruction Bad Virtual Address Register	R
MMU_DBVAR	14	Data Bad Virtual Address Register	R
MMU_PARAMS	15	Caches & TLBs hardware parameters	R
MMU_RELEASE	16	Generic MMU release number	R

Instruction request

Instruction request, only significant if `valid` is asserted. addr must be 4-byte aligned.

```

struct InstructionRequest {
    bool valid;
    addr_t addr;
    enum ExecMode mode;
};

```

Instruction response

Valid is asserted when query has been satisfied, if no request is pending, valid is not asserted.

instruction is only valid if no error is signaled.

```

struct InstructionResponse {
    bool valid;
    bool error;
    data_t instruction;
};

```

Data request

Data request, only significant if `valid` is asserted. `addr` must be 4-byte aligned. `wdata` is only significant for be-masked bytes.

- `wdata[7:0]` is at `![addr]`, masked by `be[0]`
- `wdata[15:8]` is at `[addr+1]`, masked by `be[1]`
- `wdata[23:16]` is at `[addr+2]`, masked by `be[2]`
- `wdata[31:24]` is at `[addr+3]`, masked by `be[3]`

When type is `XTN_READ` or `XTN_WRITE`, `addr` must be an opcode of enum `ExternalAccessType`. For extended access types needing an address, address is passed through the `wdata` field.

```
struct DataRequest {
    bool valid;
    addr_t addr;
    data_t wdata;
    enum DataOperationType type;
    be_t be;
    enum ExecMode mode;
};
```

Data response

Valid is asserted when query has been satisfied, if no request is pending, valid is not asserted.

data is only valid if no error is signaled.

Read data is aligned with the same semantics than the `wdata` field in struct `DataRequest`. Only bytes asserted in the BE field upon request are meaningful, others have an undefined value, they may be non-zero.

```
struct DataResponse {
    bool valid;
    bool error;
    data_t rdata;
};
```

Functions

void reset()

Reset processor internal register. Iss must behave like the processor receiving a reset cycle.

void getRequests(struct InstructionRequest &, struct DataRequest &)

Iss must populate the request fields.

uint32_t executeNCycles(uint32_t ncycle, const struct InstructionResponse &, const struct DataResponse &, uint32_t irq_bit_field)

Tell the Iss to execute *at most* `ncycle` cycles, knowing the values of the responses and the value of all the irq lines. The responses may not be valid. Each irq is a bit in the `irq_bit_field` word.

Iss must return the number of cycles it actually executed, knowing the inputs (responses and irqs) won't change. This number is at most `ncycle`. The `ncycle` argument may be 0 if wrapper only wants the ISS to handle its inputs, but not actually change the processor state. This is mostly used on GDB breakpoints.

bool virtualToPhys(addr_t &addr) const

Iss translate virtual address to physical address if it the processor contains an internal MMU. It returns false if the virtual address is not mapped. This function does nothing but returning true if no MMU is implemented in the Iss.

void setWriteBerr()

The cache received an imprecise write error condition, this signalling is asynchronous.

Other APIs

Sideband signals

In order to inform the ISS about some cache characteristics, those functions have been defined.

Their implementation is optional.

void setCacheInfo(const struct CacheInfo &info)

Informs the Iss about the cache characteristics. New fields could be added in the `Iss2::CacheInfo` definition. Current definition is:

```
struct CacheInfo
{
    bool has_mmu;
    size_t icache_line_size;
    size_t icache_assoc;
    size_t icache_n_lines;
    size_t dcache_line_size;
    size_t dcache_assoc;
    size_t dcache_n_lines;
};
```

This function supersedes the two following deprecated ones.

For backwards compatibility, default implementation of `setCacheInfo()` calls `setICacheInfo` and `setDCacheInfo`.

void setICacheInfo(size_t line_size, size_t assoc, size_t n_lines) **[deprecated]**

Inform the Iss about the instruction cache characteristics

void setDCacheInfo(size_t line_size, size_t assoc, size_t n_lines) **[deprecated]**

Inform the Iss about the data cache characteristics

Debugger API

This API is optional, it serves to expose the internal ISS registers to a debugger.

The debugger API is ISS-architecture independant.

unsigned int debugGetRegisterCount()

Iss must return the count of registers known to GDB. This must follow GDB protocol for this architecture.

debug_register_t debugGetRegisterValue(unsigned int reg)

Accessor for an Iss register, register number meaning is defined in GDB protocol for this architecture.

void debugSetRegisterValue(unsigned int reg, debug_register_t value)

Accessor for an Iss register, register number meaning is defined in GDB protocol for this architecture.

size_t debugGetRegisterSize(unsigned int reg)

Get the size for a given register. This is defined in GDB protocol for this architecture.

void dump()

Dumps internal state of the ISS on std::cout. This is used by instrumentation tools which want to display the state of an ISS at a certain event.

void set_debug_mask() and the m_debug_mask variable

set_debug_mask() is a public method for enabling processor-specific debug messages. 0 always mean "all debug messages are disabled", other values a processor-specific. Processor implementations may access the m_debug_mask member to conditionally enable debug messages.

Implementation notes

executeNCycles semantics

When executeNCycles is called, instruction and data requests previously retrieved through getRequests() may not be satisfied yet.

As executeNCycles ensures responses **MUST NOT** change for at least ncycle:

- an ISS frozen for a Data miss **MAY** continue to fetch Instructions
- an ISS frozen for an Instruction miss **MAY** continue to do Data accesses
- an ISS frozen for a Data miss **MUST** not change Data access until satisfied
- an ISS frozen for a Instruction miss **MAY** change Instruction request (if receiving an IRQ and jumping to ISR while stalled, for instance)
- an ISS running because all its instruction and data accesses are satisfied **SHOULD** run as long as no other request needs to be answered by cache.