Specifications document
Traffic Injector

May 2021

1 Injector Specifications

1.1 Module description

This unit acts as an AHB/AXI Master IP connected to the main bus on the SELENE Platform. It acts as a core with limited capabilities, only generating transactions to the bus by reading and writing to the AHB Slave RAM memory and controlled via APB registers.
The injector works along with the multi-core setup instantiated on the platform and other peripherals and monitoring units.
The module's specifications described in this section include non-implemented features, and it might be revisited for future improvements.
In order to generate traffic to the bus, the module performs a set of data transactions based on descriptors set at startup into a predefined memory address range.
The Traffic Injector is based on generic Direct Memory Access functionality fundamentals and extends its features to meet the injector objectives.
The internal components and respective configuration and functionalities are described in next sections.

1.2 Operation

1.2.1 Overview

When the injector execution is enabled by setting the **EN** bit in the **Control APB Register**, descriptor execution starts from the first descriptor, which is pointed by the **Descriptor Pointer APB Register**. The injector decodes the descriptor configuration, identifies the type of transaction, and continues with the execution. On completion of each descriptor, an interrupt flag may be configured and then continues with the **next descriptor pointer** set by the current descriptor's next descriptor pointer field value. Any disabled descriptor (**EN** bit is zero in the d**escriptor control word**) will be skipped.
The injector should be configured such that the last descriptor has the **next.last** bit set to 1 (inside the **next descriptor pointer APB register**). After completing the last descriptor on the queue, the injector will remain **idle**.
All descriptors on the queue are read at startup and loaded into the internal FIFO. After that last descriptor read, the first descriptor in the FIFO will get executed and will follow the predefined order.
In the case of configuring the Queue Mode bit (QMode in the Control APB Register), the **next.last** bit is ignored, the last descriptor in the queue will execute the first descriptor, in a loop manner. Once the **EN** bit is cleared, the injector will stop its execution.

1.3 Descriptors specification

Descriptors are used to define, control, and monitor transactions in the Traffic Injector. Descriptor types supported by this module can be classified as **read**, **write** and **delay** descriptors. Furthermore, each transaction type has the possibility of starting a **burst transfer** by **not fixing** the Source and Destinations bits in the **Descriptor Control Word**.

1.3.1 Descriptor format

A single descriptor uses 20 Bytes of memory to be configured and monitored correctly.

Address Offset	Field
0x00	Control Word
0x04	Next descriptor pointer
0x08	Destination base address
0x0C	Source base address
0x10	Status word

Table 1: Descriptor fields for configuration

On a general perspective, Figure 1 shows the main block structure on the importance and flexibility that the descriptor system gives. Later on, we are going to describe each of the mentioned parameters in the diagram.

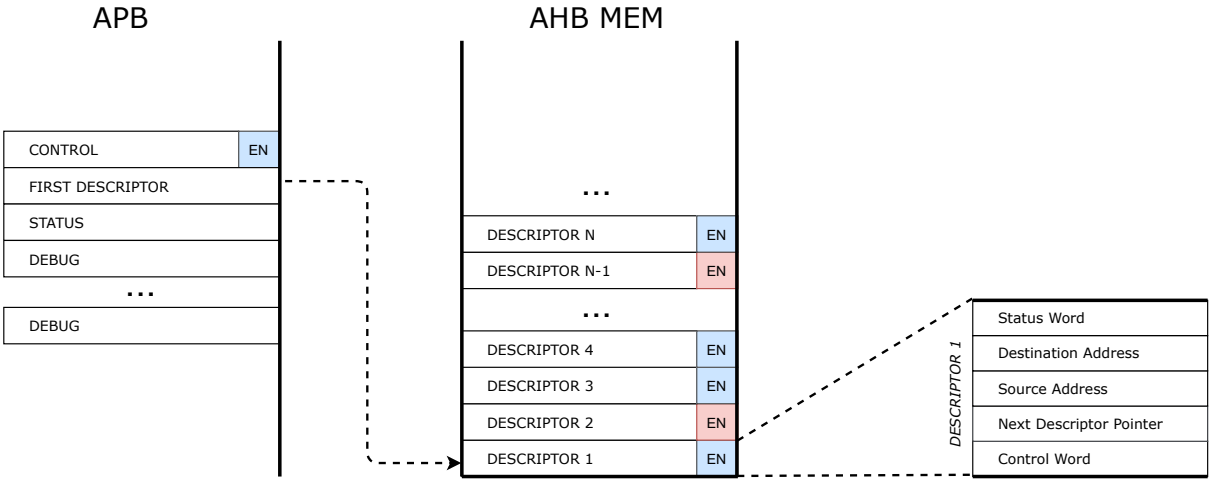
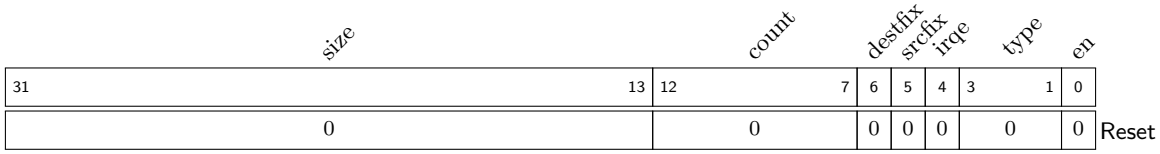


Figure 1: Injector's descriptors structure. They are controlled from the APB Registers and written to the platform's main memory.

Control word The control word configures the main parameters of the descriptor.

Register 1.1: Data descriptor control word (ctrl - 0x00)



size Total size of data to be transferred from source to destination. Each bit defines a byte that is going to be sent/received. The minimum size is 4 bytes (A full address on a 32-bit address bus configuration).

count Number of transaction repetitions.

dstfix Flag: All data is to be written from the same (fixed) destination address.

srcfix Flag: All data is to be read from the same (fixed) source address. to a second data line. When this bit is 0, a second line is available.

irqe Enable interrupt on descriptor completion.

type Descriptor type

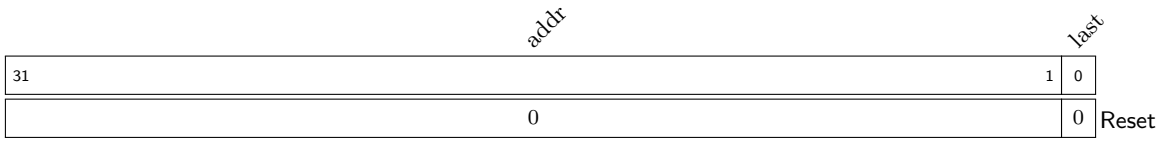
- 0: Read descriptor
- 1: Write descriptor
- 2: Delay descriptor

en Enable data descriptor

- 0: Disabled
- 1: Enabled

Next descriptor pointer It indicates the address for the next descriptor that has been set up. If the current descriptor is the last one in the queue, the **last** bit has to be '1'.

Register 1.2: Next descriptor pointer (next - 0x04)



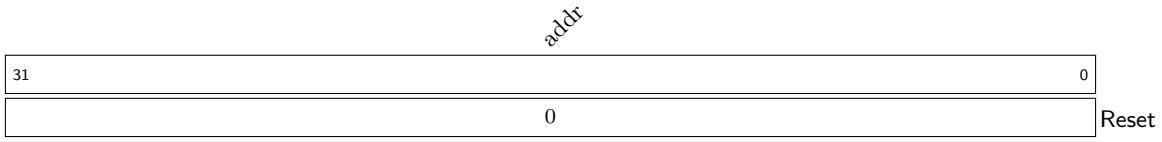
addr MSB of next descriptor start address.

last Last descriptor in the descriptor queue.

- 0: Not last descriptor.
- 1: Last descriptor.

Destination address It is used to indicate the destination address for the descriptor transaction. That means that only if we configure a **write** descriptor, this parameter will become relevant.

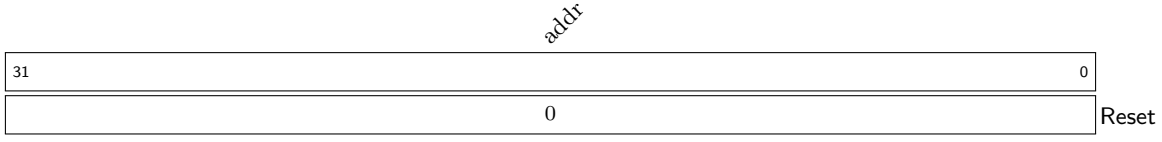
Register 1.3: Destination base address (dest - 0x08)



addr Destination base address to which data is to be written.

Source address Is used to indicate the source address for the descriptor transaction. That means that only if we configure a **read** descriptor, this parameter will become relevant.

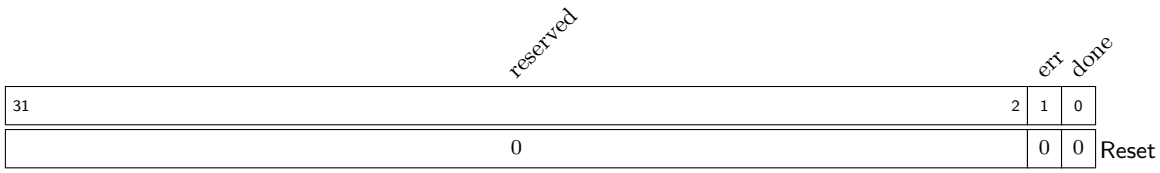
Register 1.4: Source base address (src - 0x0C)



addr Source base address to which data is to be written.

Status word The descriptor status currently shows only if an error has occurred or if the transaction has been done. The injector status and the possible error flag gets propagated to the **Status APB Register.**

Register 1.5: Descriptor status word (sts - 0x10)



err Descriptor execution error status.

- 0: No error.
- 1: Error during execution.

done Descriptor completion without error.

- 0: Not completed.
- 1: Completed.

1.4 Features

1.4.1 Interrupts

The module provides interrupt on error and interrupt on descriptor completion. The general interrupt flag is controlled via de **Control APB Register**. The Error Interrupt can also be enabled from the same register. The On-Completion Interrupt is configured in the **Descriptor Control Word**.

1.4.2 Error handling

The module provides a very straightforward way to detect and debug errors while enabled. The **Status APB register** has a general error flag that is raised if a miss-behavior has been detected.

- **DE**: `Decode Error` can occur during the decoding of a descriptor if the type of descriptor is not valid.
- **RE**: `Read Descriptor Error` will be flagged in the case when the Master Bus I/F receives an error response during a descriptor read transaction.
- **RDE**: If the Bus Master I/F receives an error during any part of the read access performed as part of the `receiver to sender`, `RDE` will be flagged.
- **WDE**: If the Bus Master I/F receives an error during any part of the write access performed as part of the `sender to receiver` transaction, `WDE` will be flagged.
- **NPE**: Defines the `Next Pointer Error` when the Bus Master I/F receives an error while switching to the next descriptor address.

1.4.3 Status monitoring

The injector provides a 5-bit Status (ST) field, which always displays the descriptor's current state. In case of an error, the **ST** field will freeze at the exact state where the error occurred, enabling the user to debug what happened.
During a transaction, the **ONG** bit in the **Status APB Register** will be set to one. In case of an error, or if the injector has completed the execution of the entire queue, it will stay on an Idle state and clear the **ONG** bit. The complete (**CMP**) bit will be set to one if no errors occurred and the descriptors queue is done.
The **Descriptor pointer debug capability APB register** shows the base address where the current descriptor was read from.

1.4.4 Pause and resume

If the **EN** bit is cleared during execution, the injector (after it has completed the ongoing descriptor) will pause by setting the **PAU** bit and clearing the **ONG** bit. The module will stay idle until **EN** and **KICK** are set to '1' in the **Control APB register**.
This feature is not implemented in the current version of the injector. Instead of pausing, the user shall restart (**RST** bit) and disable the module (set the **EN** bit to '0') to stop the execution.

1.4.5 Descriptors

As described in the **Descriptor specifications**, this module provides a simple way of configuring bus transactions so that it is possible to customize the type of transaction, addresses, repetitions, etc. A descriptor is a set of configuration registers that encapsulates all this functionality. The module is capable of decoding descriptors and sequentially executing them.

1.4.6 Transaction repetition mode

One of the advanced functionalities is the use of transaction repetitions. This enables the execution of one type transaction bursts by configuring its size and number of repetitions.
The module can also be configured with a circular queue behavior. The descriptors queue will be continuously executed until the **EN** bit is cleared from the APB Control Register. This circular behavior is accomplished with the **QMode** bit.

1.4.7 Transaction Queue

Inside the System's Platform Memory, the user should define a range of addresses reserved for instantiating all the descriptors. When the module is enabled, the first descriptor will be fetched into an internal FIFO. Once all the descriptors are read, the FIFO will be enabled and executed. The module will iteratively increment its address to fetch the next descriptor.
As we have described earlier, the last descriptor on the queue has to include the **next.last** bit for a correct operation.

1.5 Top level module

The top-level module implements an AMBA AHB wrapper with an APB and AHBM bus ports. The interface signals of the module are seen in Table 2. A future revision of the module will include more portability to other Bus Protocols.

Signal Name	Type	Function	Active
clk	INPUT	System clock signal	Low
rstn	INPUT	Reset signal	Low
APBI	INPUT	APB slave input port	-
APBO	OUTPUT	APB slave output port	-
AHBMI	INPUT	AHB Master input port	-
AHBMO	OUTPUT	AHB Master output port	-

Table 2: Injector top level signal ports

The injector *VHDL generics* configuration is shown in Table 3. Note that most of these generics are configured following the GRLIB IP Cobham Gaisler Module.

Generic name	Function	Allowed range	Default
hindex	AHB master index	0 - NAHBMST-1	0
pindex	APB slave index	0 - NAPBSLV - 1	0
paddr	Address field of the APB interface	0 - 16#FFF#	0
pmask	Mask field of the APB	0 - 16#FFF#	16#FFF#
pirq	Interrupt line used by the Injector	0 - NAHBIRQ-1	1
abits	Address bits for the descriptors queue	0 - 10	4
dbits	Bus master front end data width	32 - 128	32
max_burst_length	Maximum burst length	2 - 128	512
fifo_len	FIFO length	2 - 16	8

Table 3: Injector top level configuration parameters

The next sections will describe how the module behaves internally. Figure 2 introduces the interconnection signals used between sub-modules as a graphical visualization of how the control signals propagate.

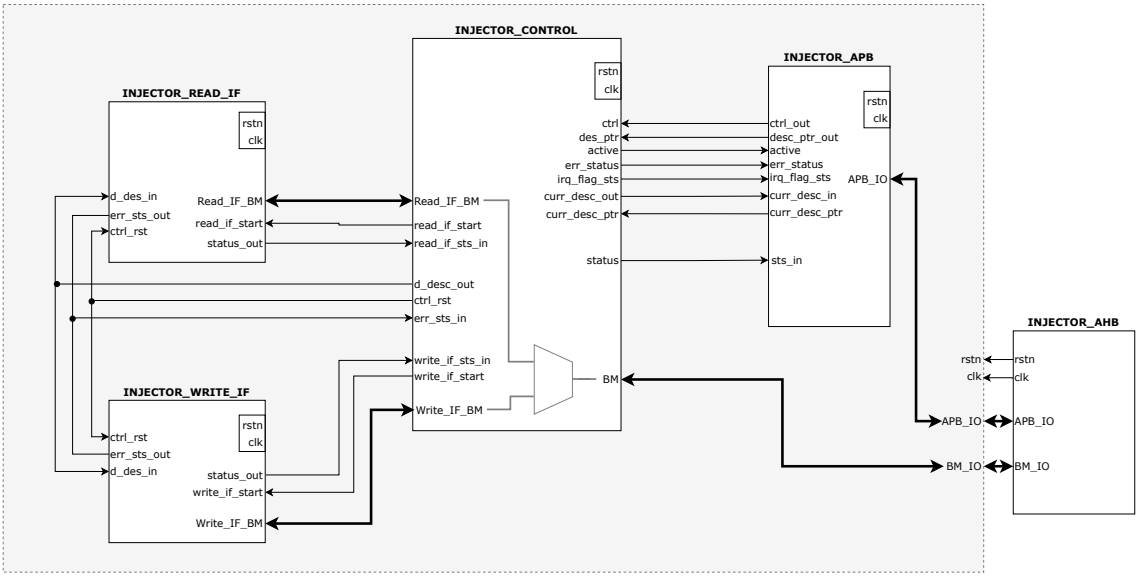


Figure 2: Module internal block diagram and entity ports

1.6 Subsystem modules

1.6.1 AHB Master Interface

The injector has an AHB master port interfacing between the system bus and the control unit signals.
Bus master interface front end data width can be configured to 32, 64, or 128 bits based on the VHDL generic `dbits`. Burst accesses are always limited by a 1KB boundary as specified in AMBA 2.0. Moreover, in De-RISC and SELENE, the Bus is configured for up to 512B bursts. When a burst finishes, or if the descriptor is set up above the permitted burst limit, the burst will be interrupted at the specified memory boundary. An idle cycle will be inserted making re-arbitration on the AHB bus more frequent.

1.6.2 APB Slave Interface

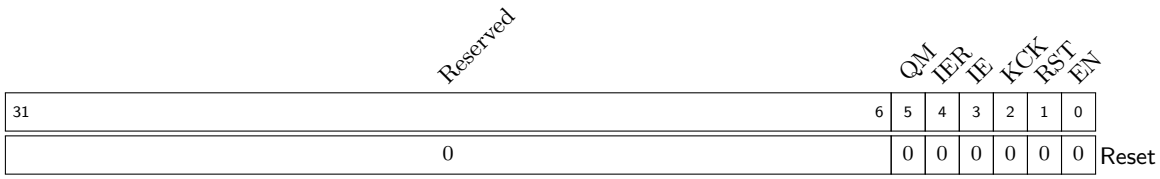
Register specifications The traffic injector is controlled and monitored through registers mapped into a defined APB address space using the VHDL generics `paddr` and `pmask` as described in Table 3.
The current injector version implements the shown registers in Table 4. Note how descriptors ranging from `0x10` up to `0x24` are used for debug capabilities and could be disabled to reduce the module footprint without losing any feature.

APB Address Offset	Register
0x00	Control Register
0x04	Status Register
0x08	First descriptor pointer
0x0C	Future capabilities register
0x10	**Debug** Descriptor control word
0x14	**Debug** Next descriptor pointer
0x18	**Debug** Destination address
0x1C	**Debug** Source address
0x20	**Debug** Descriptor status
0x24	**Debug** Current descriptor pointer

Table 4: APB register map

Control The Control register allows the configuration of interrupt generation on error and descriptor completion. There are designated bits which allows different functionalities like kicking, enabling, resetting and enabling the QMode.

Register 1.6: Control register (CTRL - 0x00)



QM Queue mode:

- 0: Sequential queue
- 1: Circular queue

IER Enable interrupt on error events.

IE

- 0: Disable interrupt
- 1: Enable interrupt

KCK Kick Injector. Reads the current descriptor next address again (if an error occurs or new descriptor is added to the completed queue).

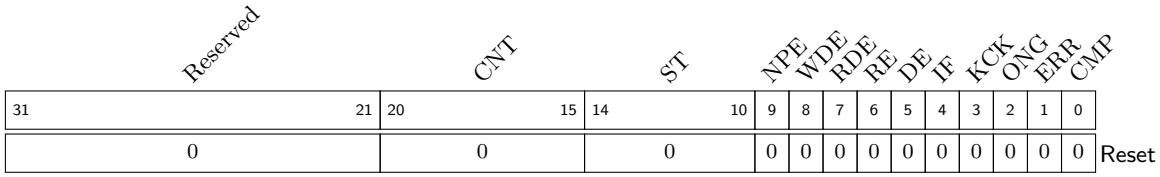
RST Reset injector. Setting this bit to one resets the core completely.

EN Enable Injector controller

- 0: Disabled
- 1: Enabled

Status Display of any error specified in 1.3, current injector status from its internal FSM, and completion monitoring.

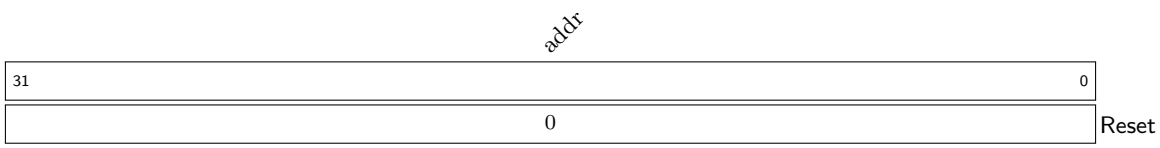
Register 1.7: Status register (STS - 0x04)



CNT	Current transaction repetition count value.
ST	Current Injector operation state.
NPE	Error while reading next descriptor pointer register.
WDE	Error while witing data during Control to Master transaction. Write Interface error.
RDE	Error while reading data during Master to Control transaction. Read Interface error.
RE	Read descriptor error.
DE	Decode descriptor error.
IF	Interrupt flag.
KCK	Kick flag pending to be executed.
ONG	Ongoing descriptor queue execution.
ERR	Error flag during descriptor queue execution.
CMP	Completed execution of the descriptor queue.

First descriptor pointer Defines the first address where the first descriptor is allocated.

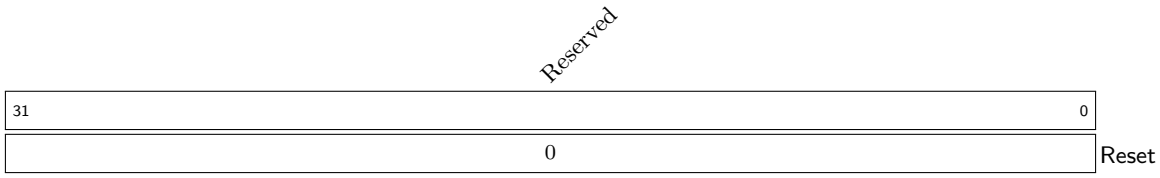
Register 1.8: First descriptor pointer (FPTR - 0x08)



addr	First descriptor pointer register. Points to the first descriptor of the queue.

Future capabilities It is a reserved register that has already been specified for future module releases. Just like the **Debug APB Registers**, this register can be disabled.

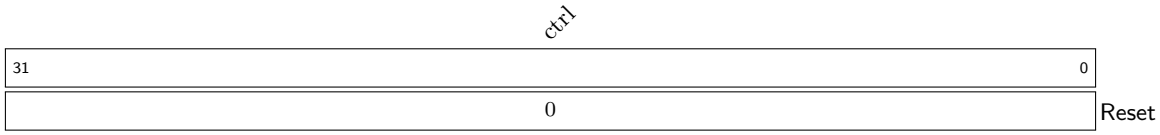
Register 1.9: Future capabilities register (FCPB - 0x0C)



Reserved This register is reserved for future features.

Descriptor control word for debug capability .

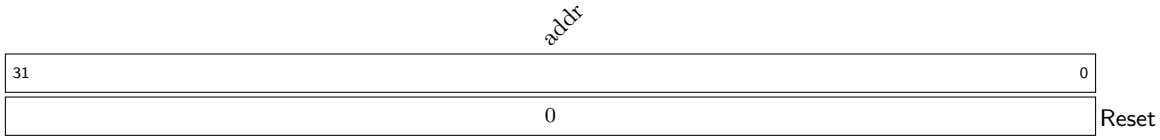
Register 1.10: Descriptor control word for debug capability (DCTR - 0x010)



ctrl Current descriptor's control field for debug capability.

Next descriptor pointer for debug capability .

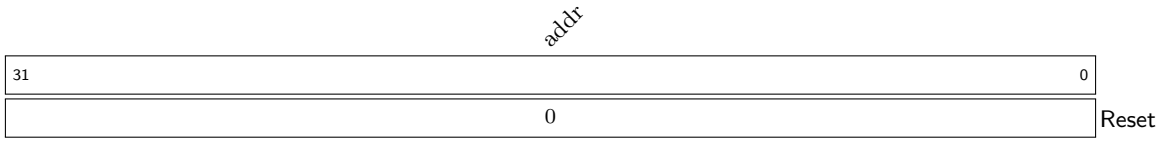
Register 1.11: Next descriptor pointer for debug capability (DNXT - 0x014)



addr Next descriptor field for debug capability.

Destination address for debug capability .

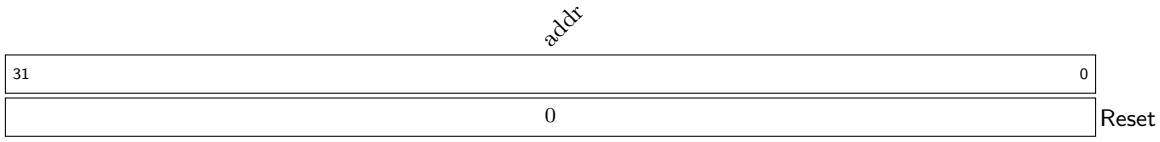
Register 1.12: Destination address for debug capability (DDST - 0x018)



addr Current descriptor's destination address field for debug capability.

Source address for debug capability .

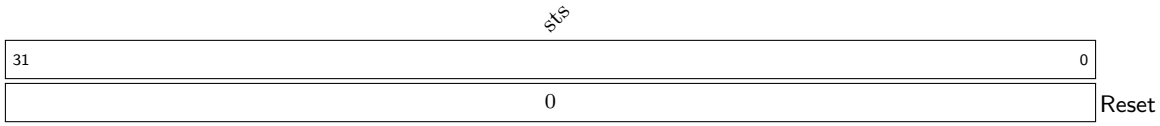
Register 1.13: Source address for debug capability (DSRC - 0x01C)



addr Current descriptor's source address field for debug capability.

Descriptor descriptor status for debug capability .

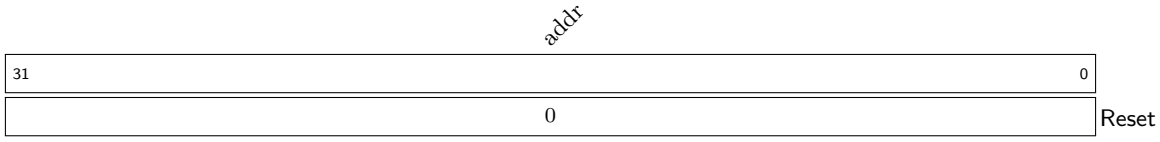
Register 1.14: Descriptor status for debug capability (DSTS - 0x020)



sts Current descriptor's status word for debug capability.

Current descriptor pointer for debug capability .

Register 1.15: Current descriptor pointer debug capability (DPTR - 0x024)



addr Pointer from which the current descriptor is read, for debug capability.

1.6.3 Control Unit

The control unit is in charge of decoding descriptors and routes the transaction requests to the respective sub-modules (either take control of the **Read Interface**, **Write Interface** or **Delay Interface**). Also, it is used to monitor the status and errors of these transactions and report them back to the APB Interface. A finite state machine is implemented in order to achieve this functionality.

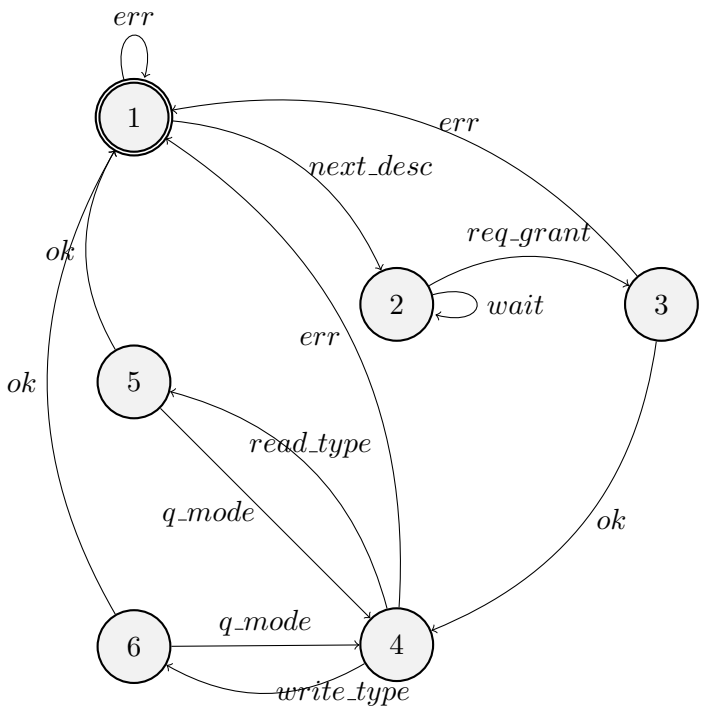


Figure 3: Control unit FSM diagram

Description

1. Idle Execution starts from Idle state and comes back after completion of each descriptor write to FIFO and on FIFO completion. The core processes a descriptor only if no errors have been encountered and if the core is enabled. If the execution is ongoing and is not the last descriptor, the core proceeds with the next descriptor fetch. Else if the whole queue is completed, execution is paused, and the core stays idle. If the core is disabled or an error occurred, the Injector goes into a disabled state and shows the corresponding error flag.

2. Fetch Descriptor Initiates a 20 Byte long burst read through the Bus Master Interface to read the descriptor fields.

3. Read Descriptor Once they are read, we save the descriptor fields to the corresponding FIFO position to be executed later on.

4. Read FIFO When all descriptors are loaded into the FIFO or the FIFO is full, the first position is read and stored in internal descriptor registers for decoding. Once all the descriptors are executed, execution goes back to Idle state and marks the completion flags.

5. Decode Descriptor Checks if the descriptor is enabled. Disabled descriptors are skipped and core jumps to read FIFO to proceed with the next descriptor in the queue.
If the descriptor is enabled, based on the `desc_type` field value, it decodes the type of the descriptor and jumps to respective states:

- **Read descriptor type**: It sends `read_if_start` signal to the `injector_read_if` sub-module.
- **Write descriptor type**: It sends a `write_if_start` signal to the `injector_write_if` sub-module.
- **Delay descriptor type**: It sends a `delay_if_start` signal to the `injector_delay_if` sub-module.

6. Read I/F Always monitors for any errors and status from `injector_read_if` sub-module. If an error is reported from the read interface sub-module, it handles the error. When a `read_if_comp` status is received, the core checks for the **Repetition count value**, it jumps to state five and starts the same descriptor again. If not, the core jumps to read FIFO state and sends a completed signal..

7. Write I/F Always monitors for any errors and status from `injector_write_if` module. If an error is reported from its interface sub-module, it handles the error. When a `write_if_comp` status is received, the core checks for the **Repetition count value**, it jumps to state five and starts the same descriptor again. If not, the core jumps to read FIFO state and sends a completed signal.

7. Delay I/F Always monitors for any errors and status from `injector_delay_if` module. If an error is reported from its interface sub-module, it handles the error. When a `delay_if_comp` status is received, the core checks for the **Repetition count value**, it jumps to state five and starts the same descriptor again. If not, the core jumps to read FIFO state and sends a completed signal.

1.6.4 Read Interface Module

The Injector Read Interface deals with data fetch from memory. The descriptor fields are passed from the injector control sub-module.
The Read Interface will continue executing until the transaction size specified in the current descriptor size field is completely transferred.

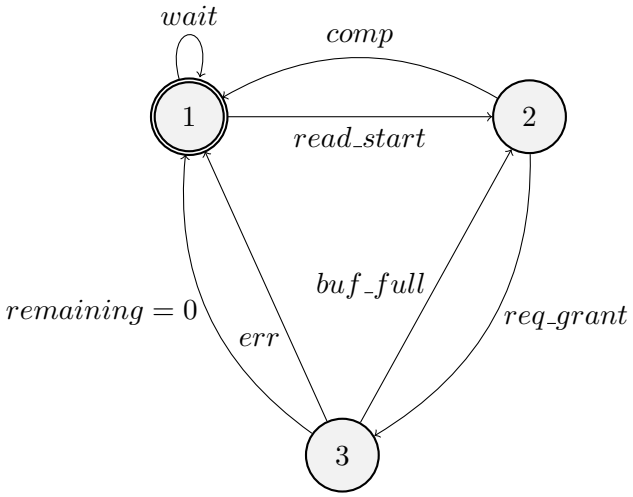


Figure 4: Read Interface FSM diagram

Description

1. Idle Its the starting state. Waits for `read_start_in` signal to proceed. Decides what size of the current burst is to be transferred depending on the descriptor parameters.

2. Execute Data Descriptor Initiates a bus master interface burst read transfer with the specified source address and transaction size.

3. Read Data Reads each data word from the bus master interface output until the total data is transferred.

1.6.5 Write interface module

The Write Interface module deals with data write transactions to AHB Addresses.
Data descriptor fields are passed from the control sub-module, and `write_if_start` signal is asserted only for enabled descriptors. Once the dummy data (a 32-bit array of 1's) is completely transferred to memory according to the specified size in the descriptor control word field, execution stops the Write Interface operation. It goes back to the main control state machine in the injector control.

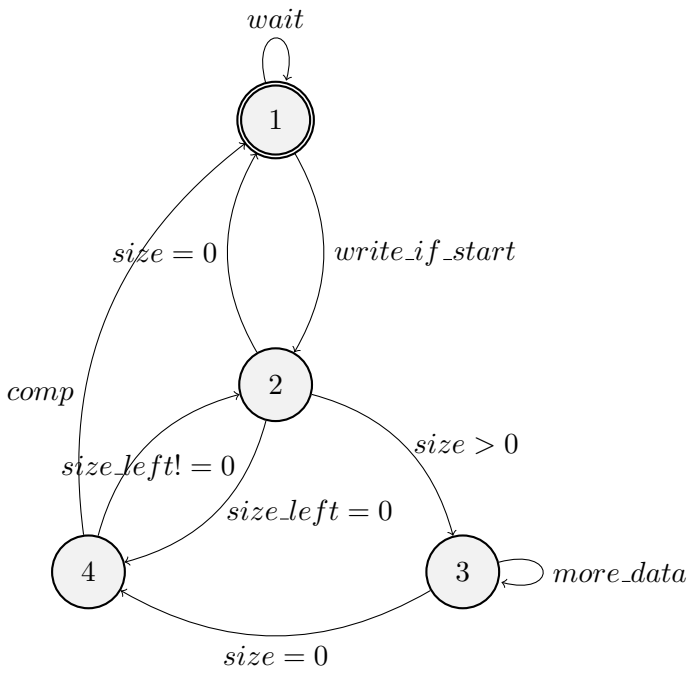


Figure 5: Write Interface FSM diagram

Description

1. Idle Starting state, waits for `write_if_start` signal from the control sub-module to proceed.

2. First Word If there is data to be written, the FSM proceeds to fetch and compute the transfer's addresses and sizes and starts a normal 4 byte aligned or burst transfer.

3. Write Burst If we still have data after a first 4-byte transfer, a burst transaction starts and loops until finished.

4. Write Data Check Once a normal transaction or burst transaction ends, we check if we need to start another burst (we have to keep in mind the upper bound on the permitted burst size) or the transaction has finished.

1.6.6 Delay Interface Module

The Injector Delay Interface transaction delays between executions. The descriptor fields are passed from the injector control sub-module.
The Delay Interface will continue executing until the transaction size (delay size) specified in the descriptor size field is completed.

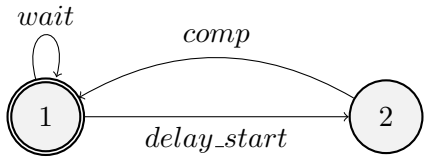


Figure 6: Delay Interface FSM diagram

Description

1. Idle Its the starting state. Waits for `delay_start_in` signal to proceed. Decides what size of the current delay is to be executed depending on the descriptor parameters.

2. Execute Descriptor Initiates a clock counter countdown with the specified size. Returns to Idle when finished.