14. On-Chip FIFO Memory Core

QII55002-9.1.0

Core Overview

The on-chip FIFO memory core is a configurable component used to buffer data and provide flow control in an SOPC Builder system. The FIFO can operate with a single clock or with separate clocks for the input and output ports. The on-chip FIFO memory core does not support burst read or write.

The input interface to the FIFO may be an Avalon® Memory Mapped (Avalon-MM) write slave or an Avalon Streaming (Avalon-ST) sink. The output interface can be an Avalon-ST source or an Avalon-MM read slave. The data is delivered to the output interface in the same order that it was received at the input interface, regardless of the value of channel, packet, frame, or any other signals.

In single clock mode, the on-chip FIFO memory includes an optional status interface that provides information about the fill-level of the FIFO. In dual clock mode, separate, optional status interfaces can be included for the input and output interfaces. The status interface also includes registers to set and control interrupts.

The on-chip FIFO memory core is SOPC Builder-ready and integrates easily into any SOPC Builder-generated system. Device drivers are provided in the HAL system library allowing software to access the core using ANSI C.

This chapter contains the following sections:

Functional Description

The on-chip FIFO memory has four configurations:

- Avalon-MM write slave to Avalon-MM read slave
- Avalon-ST sink to Avalon-ST source
- Avalon-MM write slave to Avalon-ST source
- Avalon-ST sink to Avalon-MM read slave

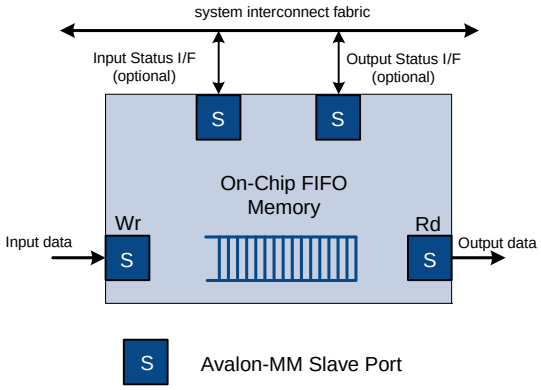
In all configurations, the input and output interfaces can use the optional backpressure signals to prevent underflow and overflow conditions. For the Avalon-MM interface, backpressure is implemented using the `waitrequest` signal. For Avalon-ST interfaces, backpressure is implemented using the `ready` and `valid` signals. For the on-chip FIFO memory, the delay between the sink asserts `ready` and the source drives valid data is one cycle.

Avalon-MM Write Slave to Avalon-MM Read Slave

In this mode, the FIFO's input is a zero-address-width Avalon-MM write slave. An Avalon-MM write master pushes data into the FIFO by writing to the input interface, and a read master (possibly the same master) pops data by reading from its output interface. The FIFO's input and output data must be the same width.

If **Allow backpressure** is turned on, the `waitrequest` signal is asserted whenever the `data_in` master tries to write to a full FIFO. `waitrequest` is only deasserted when there is enough space in the FIFO for a new transaction to complete. `waitrequest` is asserted for read operations when there is no data to be read from the FIFO, and is deasserted when the FIFO has data.

Figure 14–1. FIFO with Avalon-MM Input and Output Interfaces

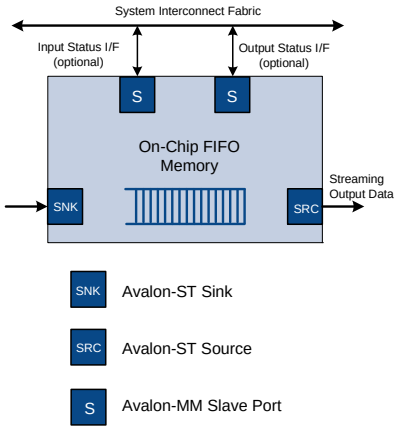


Avalon-ST Sink to Avalon-ST Source

This FIFO has streaming input and output interfaces as illustrated in Figure 14–2. You can parameterize most aspects of the Avalon-ST interfaces including the **bits per symbol**, **symbols per beat**, and the width of `error` and `channel` signals. The input and output interfaces must be the same width. If **Allow backpressure** is on in the SOPC Builder MegaWizard, both interfaces use the `ready` and `valid` signals to indicate when space is available in the FIFO and when valid data is available.

For more information about the Avalon-ST interface protocol, refer to the *Avalon Interface Specifications*.

Figure 14–2. FIFO with Avalon-ST Input and Output Interfaces

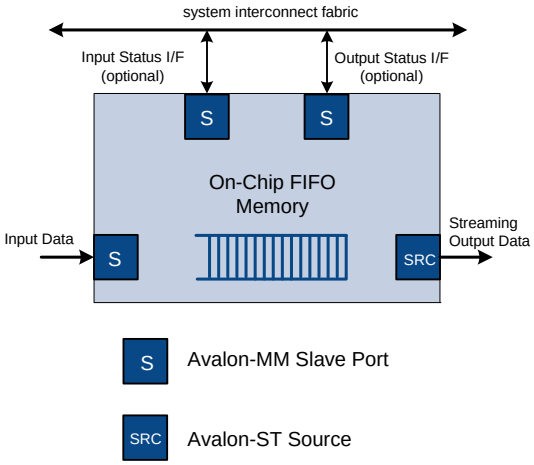


Avalon-MM Write Slave to Avalon-ST Source

In this mode, the FIFO's input is an Avalon-MM write slave with a width of 32 bits as shown in Figure 14–3. The Avalon-ST output (source) data width must also be 32 bits. You can configure output interface parameters, including: **bits per symbol**, **symbols per beat**, and the width of the `channel` and `error` signals. The FIFO performs the endian conversion to conform to the output interface protocol.

The signals that comprise this interface are mapped into bits in the Avalon's address space. If **Allow backpressure** is on, the input interface asserts `waitrequest` to indicate that the FIFO does not have enough space for the transaction to complete.

Figure 14–3. FIFO with Avalon-MM Input Interface and Avalon-ST Output Interface



The example memory map in Table 14–1 illustrates the layout of memory for a FIFO with a 32-bit Avalon-MM input interface and an Avalon-ST output interface. The output interface has 8-bit symbols, a 5-bit channel signal, and a 3-bit error signal, with packet support.

Table 14–1. Avalon-MM to Avalon-ST Memory Map

Offset	31 24	23 19	18 16	15 13	12 8	7 4	3 2	1	0
base + 0	Symbol 3	Symbol 2		Symbol 1		Symbol 0			
base + 1	reserved	reserved	error	reserved	channel	reserved	empty	EOP	SOP

If **Enable packet data** is off, the Avalon-MM write master writes all data at address offset 0 repeatedly to push data into the FIFO.

If **Enable packet data** is on, the Avalon-MM write master starts by writing the SOP, error (optional), channel (optional), EOP, and empty packet status information at address offset 1. Writing to address offset 1 does not push data into the FIFO. The Avalon-MM master then writes packet data to the FIFO repeatedly at address offset 0, pushing 8-bit symbols into the FIFO. Whenever a valid write occurs at address offset 0, the data and its respective packet information is pushed into the FIFO. Subsequent data is written at address offset 0 without the need to clear the SOP. Rewriting to address offset 1 is not required each time if the subsequent data to be pushed into the FIFO is not the end-of-packet data, as long as error and channel do not change.

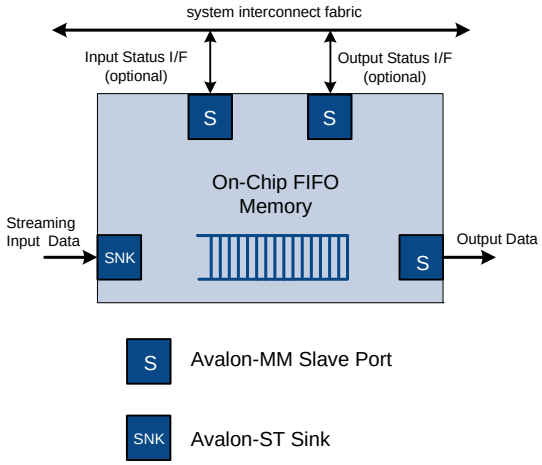
At the end of each packet, the Avalon-MM master writes to the address at offset 1 to set the EOP bit to 1, before writing the last symbol of the packet at offset 0. The write master uses the empty field to indicate the number of unused symbols at the end of the transfer. If the last packet data is not aligned with the **symbols per beat**, the empty field indicates the number of empty symbols in the last packet data. For example, if the Avalon-ST interface has **symbols per beat** of 4, and the last packet only has 3 symbols, the empty field will be 1, indicating that one symbol (the least significant symbol in the memory map) is empty.

Avalon-ST Sink to Avalon-MM Read Slave

In this mode, the FIFO's input is an Avalon-ST sink and the output is an Avalon-MM read slave with a width of 32 bits (Figure 14–4). The Avalon-ST input (sink) data width must also be 32 bits. You can configure input interface parameters, including: **bits per symbol**, **symbols per beat**, and the width of the channel and error signals. The FIFO performs the endian conversion to conform to the output interface protocol.

An Avalon-MM master reads the data from the FIFO. The signals are mapped into bits in the Avalon's address space. If **Allow backpressure** is on in the SOPC Builder MegaWizard, the input (sink) interface uses the ready and valid signals to indicate when space is available in the FIFO and when valid data is available. For the output interface, waitrequest is asserted for read operations when there is no data to be read from the FIFO. It is deasserted when the FIFO has data to send.

Figure 14–4. FIFO with Avalon-ST Input and Avalon-MM Output



As shown in Table 14–2, the memory map for the Avalon-ST to Avalon-MM slave FIFO is exactly the same as for Avalon-MM to Avalon-ST FIFO.

Table 14–2. Avalon-ST to Avalon-MM Memory Map

Offset	31 24	23 19	18 16	15 13	12 8	7 4	3	2	1	0
base + 0	Symbol 3	Symbol 2		Symbol 1		Symbol 0				
base + 1	reserved	reserved	error	reserved	channel	reserved	empty		EOP	SOP

If **Enable packet data** is off, read data repeatedly at address offset 0 to pop the data from the FIFO.

If **Enable packet data** is on, the Avalon-MM read master starts reading from address offset 0. If the read is valid, that is, the FIFO is not empty, both data and packet status information are popped from the FIFO. The packet status information is obtained by reading at address offset 1. Reading from address offset 1 does not pop data from the FIFO. The `error`, `channel`, `SOP`, `EOP` and empty fields are available at address offset 1 to determine the status of the packet data read from address offset 0.

The empty field indicates the number of empty symbols in the data field. For example, if the Avalon-ST interface has symbols-per-beat of 4, and the last packet data only has 1 symbol, then the empty field will be 3 to indicate that 3 symbols (the 3 least significant symbols in the memory map) are empty.

Status Interface

The FIFO provides two optional status interfaces, one for the master writing to the input interface and a second for the read master reading from the output interface. For FIFOs that operate in a single domain, a single status interface is sufficient to monitor the status of the FIFO. For FIFOs using a dual clocking scheme, a second status interface using the output clock is necessary to accurately monitor the status of the FIFO in both clock domains.

Clocking Modes

When single clock mode is used, the FIFO being used is SCFIFO. When dual-clock mode is chosen, the FIFO being used is DCFIFO. In dual-clock mode, input data and write-side status interfaces use the write side clock domain; the output data and read-side status interfaces use the read-side clock domain.

Device Support

The on-chip FIFO memory supports all Altera® device families except the Hardcopy® series.

Instantiating the Core in SOPC Builder

Use the MegaWizard™ interface for the on-chip FIFO memory in SOPC Builder to specify the core configuration. The following sections describe the available options.

FIFO Settings

The following sections outline the settings that pertain to the FIFO as a whole.

Depth

Depth indicates the depth of the FIFO, in Avalon-ST beats or Avalon-MM words. The default depth is 16. When dual clock mode is used, the actual FIFO depth is equal to depth-3. This is due to clock crossing and to avoid FIFO overflow.

Clock Settings

The two options are **Single clock mode** and **Dual clock mode**. In **Single clock mode**, all interface ports use the same clock. In **Dual clock mode**, input data and input side status are on the input clock domain. Output data and output side status are on the output clock domain.

Status Port

The optional status ports are Avalon-MM slaves. To include the optional input side status interface, turn on **Create status interface for input** on the SOPC Builder MegaWizard. For FIFOs whose input and output ports operate in separate clock domains, you can include a second status interface by turning on **Create status interface for output**. Turning on **Enable IRQ for status ports** adds an interrupt signal to the status ports.

FIFO Implementation

This option determines if the FIFO is built from registers or embedded memory blocks. The default is to construct the FIFO from embedded memory blocks.

Interface Parameters

The following sections outline the options for the input and output interfaces.

Input

Available input interfaces are **Avalon-MM** write slave and **Avalon-ST** sink.

Output

Available output interfaces are **Avalon-MM** read slave and **Avalon-ST** source.

Allow Backpressure

When **Allow backpressure** is on, an Avalon-MM interface includes the `waitrequest` signal which is asserted to prevent a master from writing to a full FIFO or reading from an empty FIFO. An Avalon-ST interface includes the `ready` and `valid` signals to prevent underflow and overflow conditions.

Avalon-MM Port Settings

Valid **Data widths** are 8, 16, and 32 bits.

If Avalon-MM is selected for one interface and Avalon-ST for the other, the data width is fixed at 32 bits.

The Avalon-MM interface accesses data 4 bytes at a time. For data widths other than 32 bits, be careful of potential overflow and underflow conditions.

Avalon-ST Port Settings

The following parameters allow you to specify the size and error handling of the Avalon-ST port or ports:

- **Bits per symbol**
- **Symbols per beat**
- **Channel width**
- **Error width**

If the symbol size is not a power of two, it is rounded up to the next power of two. For example, if the **bits per symbol** is 10, the symbol will be mapped to a 16-bit memory location. With 10-bit symbols, the maximum number of **symbols per beat** is two.

Enable packet data provides an option for packet transmission.

Software Programming Model

The following sections describe the software programming model for the on-chip FIFO memory core, including the register map and software declarations to access the hardware. For Nios II processor users, Altera provides HAL system library drivers that enable you to access the on-chip FIFO memory core using its HAL API.

HAL System Library Support

The Altera-provided driver implements a HAL device driver that integrates into the HAL system library for Nios II systems. HAL users should access the on-chip FIFO memory via the familiar HAL API, rather than accessing the registers directly.

Software Files

Altera provides the following software files for the on-chip FIFO memory core:

- **altera_avalon_fifo_regs.h**—This file defines the core's register map, providing symbolic constants to access the low-level hardware.
- **altera_avalon_fifo_util.h**—This file defines functions to access the on-chip FIFO memory core hardware. It provides utilities to initialize the FIFO, read and write status, enable flags and read events.
- **altera_avalon_fifo.h**—This file provides the public interface to the on-chip FIFO memory
- **altera_avalon_fifo_util.c**—This file implements the utilities listed in **altera_avalon_fifo_util.h**.

Programming with the On-Chip FIFO Memory

This section describes the low-level software constructs for manipulating the on-chip FIFO memory core hardware. Table 14–3 lists all of the available functions.

Table 14–3. On-Chip FIFO Memory Functions (Part 1 of 2)

Function Name	Description
`altera_avalon_fifo_init()`	Initializes the FIFO.
`altera_avalon_fifo_read_status()`	Returns the integer value of the specified bit of the status register. To read all of the bits at once, use the `ALTERA_AVALON_FIFO_STATUS_ALL` mask.
`altera_avalon_fifo_read_ienable()`	Returns the value of the specified bit of the interrupt enable register. To read all of the bits at once, use the `ALTERA_AVALON_FIFO_EVENT_ALL` mask.
`altera_avalon_fifo_read_almostfull()`	Returns the value of the `almostfull` register.
`altera_avalon_fifo_read_almostempty()`	Returns the value of the `almostempty` register.
`altera_avalon_fifo_read_event()`	Returns the value of the specified bit of the event register. All of the event bits can be read at once by using the `ALTERA_AVALON_FIFO_STATUS_ALL` mask.
`altera_avalon_fifo_read_level()`	Returns the fill level of the FIFO.
`altera_avalon_fifo_clear_event()`	Clears the specified bits and the event register and performs error checking.

Table 14–3. On-Chip FIFO Memory Functions (Part 2 of 2)

Function Name	Description
`altera_avalon_fifo_write_ienable()`	Writes the specified bits of the `interruptenable` register and performs error checking.
`altera_avalon_fifo_write_almostfull()`	Writes the specified value to the `almostfull` register and performs error checking.
`altera_avalon_fifo_write_almostempty()`	Writes the specified value to the `almostempty` register and performs error checking.
`altera_avalon_fifo_write_fifo()`	Writes the specified data to the `write_address`.
`altera_avalon_fifo_write_other_info()`	Writes the packet status information to the `write_address`. Only valid when the **Enable packet data** option is turned on.
`altera_avalon_fifo_read_fifo()`	Reads data from the specified `read_address`.
`altera_avalon_fifo_read__other_info()`	Reads the packet status information from the specified `read_address`. Only valid when the **Enable packet data** option is turned on.

Software Control

Table 14–4 provides the register map for the `status` register. The layout of `status` register for the input and output interfaces is identical.

Table 14–4. FIFO Status Register Memory Map

offset	31 24	23 16	15 8	7	6	5	4	3	2	1	0
base	`fill_level`										
base + 1						`i_status`					
base + 2						`event`					
base + 3						`interrupt enable`					
base + 4	`almostfull`										
base + 5	`almostempty`										

Table 14–5 outlines the use of the various fields of the `status` register.

Table 14–5. FIFO Status Field Descriptions (Part 1 of 2)

Field	Type	Description
`fill_level`	RO	The instantaneous fill level of the FIFO, provided in units of symbols for a FIFO with an Avalon-ST FIFO and words for an Avalon-MM FIFO.
`i_status`	RO	A 6-bit register that shows the FIFO's instantaneous status. See Table 14–6 for the meaning of each bit field.
`event`	RW1C	A 6-bit register with exactly the same fields as `i_status`. When a bit in the `i_status` register is set, the same bit in the `event` register is set. The bit in the `event` register is only cleared when software writes a 1 to that bit.
`interruptenable`	RW	A 6-bit interrupt enable register with exactly the same fields as the `event` and `i_status` registers. When a bit in the `event` register transitions from a 0 to a 1, and the corresponding bit in `interruptenable` is set, the master Is interrupted.

Table 14–5. FIFO Status Field Descriptions (Part 2 of 2)

Field	Type	Description
almostfull	RW	A threshold level used for interrupts and status. Can be written by the Avalon-MM status master at any time. The default threshold value for DCFIFO is Depth-4. The default threshold value for SCFIFO is Depth-1. The valid range of the threshold value is from 1 to the default. 1 is used when attempting to write a value smaller than 1. The default is used when attempting to write a value larger than the default.
almostempty	RW	A threshold level used for interrupts and status. Can be written by the Avalon-MM status master at any time. The default threshold value for DCFIFO is 1. The default threshold value for SCFIFO is 1. The valid range of the threshold value is from 1 to the maximum allowable almostfull threshold. 1 is used when attempting to write a value smaller than 1. The maximum allowable is used when attempting to write a value larger than the maximum allowable.

Table 14–6 describes the instantaneous status bits.

Table 14–6. Status Bit Field Descriptions

Bit(s)	Name	Description
0	FULL	Has a value of 1 if the FIFO is currently full.
1	EMPTY	Has a value of 1 if the FIFO is currently empty.
2	ALMOSTFULL	Has a value of 1 if the fill level of the FIFO is greater than the almostfull value.
3	ALMOSTEMPTY	Has a value of 1 if the fill level of the FIFO is less than the almostempty value.
4	OVERFLOW	Is set to 1 for 1 cycle every time the FIFO overflows. The FIFO overflows when an Avalon write master writes to a full FIFO. OVERFLOW is only valid when **Allow backpressure** is off.
5	UNDERFLOW	Is set to 1 for 1 cycle every time the FIFO underflows. The FIFO underflows when an Avalon read master reads from an empty FIFO. UNDERFLOW is only valid when **Allow backpressure** is off.

Table 14–7 lists the bit fields of the event register. These fields are identical to those in the status register and are set at the same time; however, these fields are only cleared when software writes a one to clear (W1C). The event fields can be used to determine if a particular event has occurred.

Table 14–7. Event Bit Field Descriptions

Bit(s)	Name	Description
1	E_FULL	Has a value of 1 if the FIFO has been full and the bit has not been cleared by software.
0	E_EMPTY	Has a value of 1 if the FIFO has been empty and the bit has not been cleared by software.
3	E_ALMOSTFULL	Has a value of 1 if the fill level of the FIFO has been greater than the almostfull threshold value and the bit has not been cleared by software.
2	E_ALMOSTEMPTY	Has a value of 1 if the fill level of the FIFO has been less than the almostempty value and the bit has not been cleared by software.
4	E_OVERFLOW	Has a value of 1 if the FIFO has overflowed and the bit has not been cleared by software.
5	E_UNDERFLOW	Has a value of 1 if the FIFO has underflowed and the bit has not been cleared by software.

Table 14–8 provides a mask for the six STATUS fields. When a bit in the event register transitions from a zero to a one, and the corresponding bit in the interruptenable register is set, the master is interrupted.

Table 14–8. InterruptEnable Bit Field Descriptions

Bit(s)	Name	Description
1	IE_FULL	Enables an interrupt if the FIFO is currently full.
0	IE_EMPTY	Enables an interrupt if the FIFO is currently empty.
3	IE_ALMOSTFULL	Enables an interrupt if the fill level of the FIFO is greater than the value of the `almostfull` register.
2	IE_ALMOSTEMPTY	Enables an interrupt if the fill level of the FIFO is less than the value of the `almostempty` register.
4	IE_OVERFLOW	Enables an interrupt if the FIFO overflows. The FIFO overflows when an Avalon write master writes to a full FIFO.
5	IE_UNDERFLOW	Enables an interrupt if the FIFO underflows. The FIFO underflows when an Avalon read master reads from an empty FIFO.
6	ALL	Enables all 6 status conditions to interrupt.

Macros to access all of the registers are defined in **altera_avalon_fifo_regs.h.** For example, this file includes the following macros to access the `status` register.

```
#define ALTERA_AVALON_FIFO_LEVEL_REG          0
#define ALTERA_AVALON_FIFO_STATUS_REG         1
#define ALTERA_AVALON_FIFO_EVENT_REG          2
#define ALTERA_AVALON_FIFO_IENABLE_REG        3
#define ALTERA_AVALON_FIFO_ALMOSTFULL_REG     4
#define ALTERA_AVALON_FIFO_ALMOSTEMPTY_REG    5
```

For a complete list of predefined macros and utilities to access the on-chip FIFO hardware, see:
<install_dir>\quartus\sopc_builder\components\altera_avalon_fifo\HAL\inc\altera_avalon_fifo.h and
<install_dir>\quartus\sopc_builder\components\altera_avalon_fifo\HAL\inc\altera_avalon_fifo_util.h.

Software Example

Example 14–1 shows sample codes for the core.

Example 14–1. Sample Code for the On-Chip FIFO Memory (Part 1 of 2)

```
/************************************************************************/
//Includes
#include "altera_avalon_fifo_regs.h"
#include "altera_avalon_fifo_util.h"
#include "system.h"
#include "sys/alt_irq.h"
#include <stdio.h>
#include <stdlib.h>

#define ALMOST_EMPTY 2
#define ALMOST_FULL OUTPUT_FIFO_OUT_FIFO_DEPTH-5

volatile int input_fifo_wrclk_irq_event;

void print_status(alt_u32 control_base_address)
{
   printf("---------------------------------------\n");
   printf("LEVEL = %u\n", altera_avalon_fifo_read_level(control_base_address) );
   printf("STATUS = %u\n", altera_avalon_fifo_read_status(control_base_address,
ALTERA_AVALON_FIFO_STATUS_ALL) );
   printf("EVENT = %u\n", altera_avalon_fifo_read_event(control_base_address,
ALTERA_AVALON_FIFO_EVENT_ALL) );
   printf("IENABLE = %u\n", altera_avalon_fifo_read_ienable(control_base_address,
ALTERA_AVALON_FIFO_IENABLE_ALL) );
   printf("ALMOSTEMPTY = %u\n",
altera_avalon_fifo_read_almostempty(control_base_address) );
   printf("ALMOSTFULL = %u\n\n",
altera_avalon_fifo_read_almostfull(control_base_address));
}

static void handle_input_fifo_wrclk_interrupts(void* context, alt_u32 id)
{
   /* Cast context to input_fifo_wrclk_irq_event's type. It is important
   * to declare this volatile to avoid unwanted compiler optimization.
   */
   volatile int* input_fifo_wrclk_irq_event_ptr = (volatile int*) context;

   /* Store the value in the FIFO's irq history register in *context. */
   *input_fifo_wrclk_irq_event_ptr =
altera_avalon_fifo_read_event(INPUT_FIFO_IN_CSR_BASE, ALTERA_AVALON_FIFO_EVENT_ALL);
   printf("Interrupt Occurs for %#x\n", INPUT_FIFO_IN_CSR_BASE);
   print_status(INPUT_FIFO_IN_CSR_BASE);

   /* Reset the FIFO's IRQ History register. */
   altera_avalon_fifo_clear_event(INPUT_FIFO_IN_CSR_BASE,
ALTERA_AVALON_FIFO_EVENT_ALL);
}

/* Initialize the fifo */
static int init_input_fifo_wrclk_control()
{
   int return_code = ALTERA_AVALON_FIFO_OK;

   /* Recast the IRQ History pointer to match the alt_irq_register() function
   * prototype. */
   void* input_fifo_wrclk_irq_event_ptr = (void*) &input_fifo_wrclk_irq_event;
   /* Enable all interrupts. */
```

Example 14–1. Sample Code for the On-Chip FIFO Memory (Part 2 of 2)

```
    /* Clear event register, set enable all irq, set almostempty and
    almostfull threshold */
    return_code = altera_avalon_fifo_init(INPUT_FIFO_IN_CSR_BASE,
                                          0, // Disabled interrupts
                                          ALMOST_EMPTY,
                                          ALMOST_FULL);
    /* Register the interrupt handler. */
    alt_irq_register( INPUT_FIFO_IN_CSR_IRQ,
input_fifo_wrclk_irq_event_ptr, handle_input_fifo_wrclk_interrupts );
    return return_code;
}
```

On-Chip FIFO Memory API

This section describes the application programming interface (API) for the on-chip FIFO memory core.

altera_avalon_fifo_init()

Prototype:	`int altera_avalon_fifo_init(alt_u32 address, alt_u32 ienable, alt_u32 emptymark, alt_u32 fullmark)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave `ienable`—the value to write to the `interruptenable` register `emptymark`—the value for the almost empty threshold level `fullmark`—the value for the almost full threshold level
Returns:	Returns 0 (`ALTERA_AVALON_FIFO_OK`) if successful, `ALTERA_AVALON_FIFO_EVENT_CLEAR_ERROR` for clear errors, `ALTERA_AVALON_FIFO_IENABLE_WRITE_ERROR` for interrupt enable write errors, `ALTERA_AVALON_FIFO_THRESHOLD_WRITE_ERROR` for errors writing the `almostfull` and `almostempty` registers.
Description:	Clears the `event` register, writes the `interruptenable` register, and sets the `almostfull` register and `almostempty` registers.

altera_avalon_fifo_read_status()

Prototype:	`int altera_avalon_fifo_read_status(alt_u32 address, alt_u32 mask)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave `mask`—masks the read value from the `status` register
Returns:	Returns the masked bits of the addressed register.
Description:	Gets the addressed register bits—the `AND` of the value of the addressed register and the mask.

altera_avalon_fifo_read_ienable()

Prototype:	`int altera_avalon_fifo_read_ienable(alt_u32 address, alt_u32 mask)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave `mask`—masks the read value from the `interruptenable` register
Returns:	Returns the logical `AND` of the `interruptenable` register and the mask.
Description:	Gets the logical `AND` of the `interruptenable` register and the mask.

altera_avalon_fifo_read_almostfull()

Prototype:	`int altera_avalon_fifo_read_almostfull(alt_u32 address)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave
Returns:	Returns the value of the `almostfull` register.
Description:	Gets the value of the `almostfull` register.

altera_avalon_fifo_read_almostempty()

Prototype:	`int altera_avalon_fifo_read_almostempty(alt_u32 address)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave
Returns:	Returns the value of the `almostempty` register.
Description:	Gets the value of the `almostempty` register.

altera_avalon_fifo_read_event()

Prototype:	`int altera_avalon_fifo_read_event(alt_u32 address, alt_u32 mask)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave `mask`—masks the read value from the `event` register
Returns:	Returns the logical `AND` of the `event` register and the mask.
Description:	Gets the logical `AND` of the `event` register and the mask. To read single bits of the event register use the single bit masks, for example: `ALTERA_AVALON_FIFO_FIFO_EVENT_F_MSK`. To read the entire `event` register use the full mask: `ALTERA_AVALON_FIFO_EVENT_ALL`.

altera_avalon_fifo_read_level()

Prototype:	`int altera_avalon_fifo_read_level(alt_u32 address)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave
Returns:	Returns the fill level of the FIFO.
Description:	Gets the fill level of the FIFO.

altera_avalon_fifo_clear_event()

Prototype:	`int altera_avalon_fifo_clear_event(alt_u32 address, alt_u32 mask)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave `mask`—the mask to use for bit-clearing (1 means clear this bit, 0 means do not clear)
Returns:	Returns 0 (`ALTERA_AVALON_FIFO_OK`) if successful, `ALTERA_AVALON_FIFO_EVENT_CLEAR_ERROR` if unsuccessful.
Description:	Clears the specified bits of the `event` register.

altera_avalon_fifo_write_ienable()

Prototype:	`int altera_avalon_fifo_write_ienable(alt_u32 address, alt_u32 mask)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`address`—the base address of the FIFO control slave `mask`—the value to write to the `interruptenable` register. See **altera_avalon_fifo_regs.h** for individual interrupt bit masks.
Returns:	Returns 0 (`ALTERA_AVALON_FIFO_OK`) if successful, `ALTERA_AVALON_FIFO_IENABLE_WRITE_ERROR` if unsuccessful.
Description:	Writes the specified bits of the `interruptenable` register.

altera_avalon_fifo_write_almostfull()

Prototype: `int altera_avalon_fifo_write_almostfull(alt_u32 address, alt_u32 data)`

Thread-safe: No.

Available from ISR: No.

Include: <**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>

Parameters: `address`—the base address of the FIFO control slave

`data`—the value for the almost full threshold level

Returns: Returns 0 (`ALTERA_AVALON_FIFO_OK`) if successful, `ALTERA_AVALON_FIFO_THRESHOLD_WRITE_ERROR` if unsuccessful.

Description: Writes `data` to the `almostfull` register.

altera_avalon_fifo_write_almostempty()

Prototype: `int altera_avalon_fifo_write_almostempty(alt_u32 address, alt_u23 data)`

Thread-safe: No.

Available from ISR: No.

Include: <**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>

Parameters: `address`—the base address of the FIFO control slave

`data`—the value for the almost empty threshold level

Returns: Returns 0 (`ALTERA_AVALON_FIFO_OK`) if successful, `ALTERA_AVALON_FIFO_THRESHOLD_WRITE_ERROR` if unsuccessful.

Description: Writes `data` to the `almostempty` register.

altera_avalon_write_fifo()

Prototype: `int altera_avalon_write_fifo(alt_u32 write_address, alt_u32 ctrl_address, alt_u32 data)`

Thread-safe: No.

Available from ISR: No.

Include: <**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>

Parameters: `write_address`—the base address of the FIFO write slave

`ctrl_address`—the base address of the FIFO control slave

`data`—the value to write to address offset 0 for Avalon-MM to Avalon-ST transfers, the value to write to the single address available for Avalon-MM to Avalon-MM transfers. See the *Avalon Interface Specifications* for the data ordering.

Returns: Returns 0 (`ALTERA_AVALON_FIFO_OK`) if successful, `ALTERA_AVALON_FIFO_FULL` if unsuccessful.

Description: Writes `data` to the specified address if the FIFO is not full.

altera_avalon_write_other_info()

Prototype:	`int altera_avalon_write_other_info(alt_u32 write_address, alt_u32 ctrl_address, alt_u32 data)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`write_address`—the base address of the FIFO write slave `ctrl_address`—the base address of the FIFO control slave `data`—the packet status information to write to address offset 1 of the Avalon interface. See the *Avalon Interface Specifications* for the ordering of the packet status information.
Returns:	Returns 0 (`ALTERA_AVALON_FIFO_OK`) if successful, `ALTERA_AVALON_FIFO_FULL` if unsuccessful.
Description:	Writes the packet status information to the `write_address`. Only valid when **Enable packet data** is on.

altera_avalon_fifo_read_fifo()

Prototype:	`int altera_avalon_fifo_read_fifo(alt_u32 read_address, alt_u32 ctrl_address)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`read_address`—the base address of the FIFO read slave `ctrl_address`—the base address of the FIFO control slave
Returns:	Returns the data from address offset 0, or 0 if the FIFO is empty.
Description:	Gets the data addressed by `read_address`.

altera_avalon_fifo_read_other_info()

Prototype:	`int altera_avalon_fifo_read_other_info(alt_u32 read_address)`
Thread-safe:	No.
Available from ISR:	No.
Include:	<**altera_avalon_fifo_regs.h**>, <**altera_avalon_fifo_utils.h**>
Parameters:	`read_address`—the base address of the FIFO read slave
Returns:	Returns the packet status information from address offset 1 of the Avalon interface. See the *Avalon Interface Specifications* for the ordering of the packet status information.
Description:	Reads the packet status information from the specified `read_address`. Only valid when **Enable packet data** is on.

Referenced Documents

This chapter references *Avalon Interface Specifications*.

Document Revision History

Table 14–9 shows the revision history for this chapter.

Table 14–9. Document Revision History

Date and Document Version	Changes Made	Summary of Changes
November 2009 v9.1.0	Added description to the core overview.	
March 2009 v9.0.0	Updated the description of the function altera_avalon_fifo_read_status().	—
November 2008 v8.1.0	Changed to 8-1/2 x 11 page size. No change to content.	—
May 2008 v8.0.0	No change from previous release.	—

For previous versions of the *Quartus II Handbook*, refer to the Quartus II Handbook Archive.