# FIFO

*User Guide*

03/2018

Hercules Microelectronics, Inc.

China

# Contents

# 1 Software FIFO Introduction

This document describes the FIFO (First-in First-out) IP core, including asynchronous FIFO and synchronous FIFO. The FIFO supports the following features:

- Support data width up to 128 bits with depth up to 1K bits;
- Support different input and output data widths;
- Support both synchronous and asynchronous FIFO;
- Support wfull/rempty status flag;
- Optional synchronous clear inputs for both read and write;
- Optional almost_full/almost_empty status flag to indicate only one more data can be written/read before FIFO is full/empty;
- Optional prog_full/prog_empty status flag. Support 4 types of programmable full flag generation : Single threshold constant, Single threshold with dedicated input port, Assert and negate threshold constants, Assert and negate thresholds with dedicated input ports;
- Optional count vector(s)(wr_data_cnt and rd_data_cnt) provide visibility into number of data words currently in the FIFO, synchronized to either clock domain;
- Four optional handshake signals (wr_ack, rd_ack, overflow, underflow) provide feedback (acknowledge ment or rejection) in response to write and read requests in the prior clock cycle;
- Invalid read or write requests are rejected without affecting the FIFO state. When FIFO is full, data can’t write to FIFO anymore, the write data will be omitted. When FIFO is empty, data can’t read out anymore;
- Support AHB interface.
- Support First Word Fall Through function
- Up to 200MHz performance.

Device Support:

HME-M5, HME -M7, HME -HR3, HME _HR2, HME _M0, HME _M1

# 2 Software FIFO Block Diagram

## 2.1 Asynchronous FIFO

Following figure shows the block diagram of asynchronous FIFO.

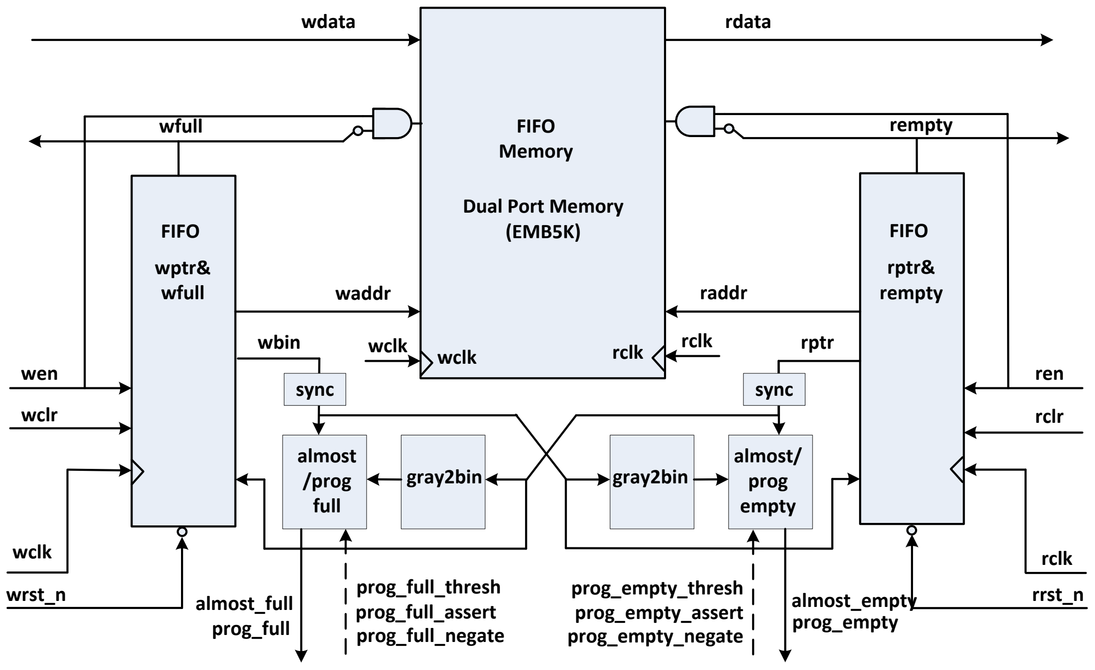

*Figure 2-1* ***Asynchronous FIFO Block Diagram***

To implement asynchronous FIFO function:

- To store the FIFO data, the dual port memory (EMB5K) are used, the number of EMB5K depends on the data width and memory depth that user need;
- Synchronous bridge between different clock domains are used to solve the metastability issue;
- To eliminate the problem associated with synchronizing multiple changing signals on the same clock edge, Gray counter is used. Gray codes only allow one bit to change for each clock transition.(Only one bit is allowed to convert at each transition clock in Gray codes).

## 2.2 Synchronous FIFO

Following figure shows the block diagram of synchronous FIFO:

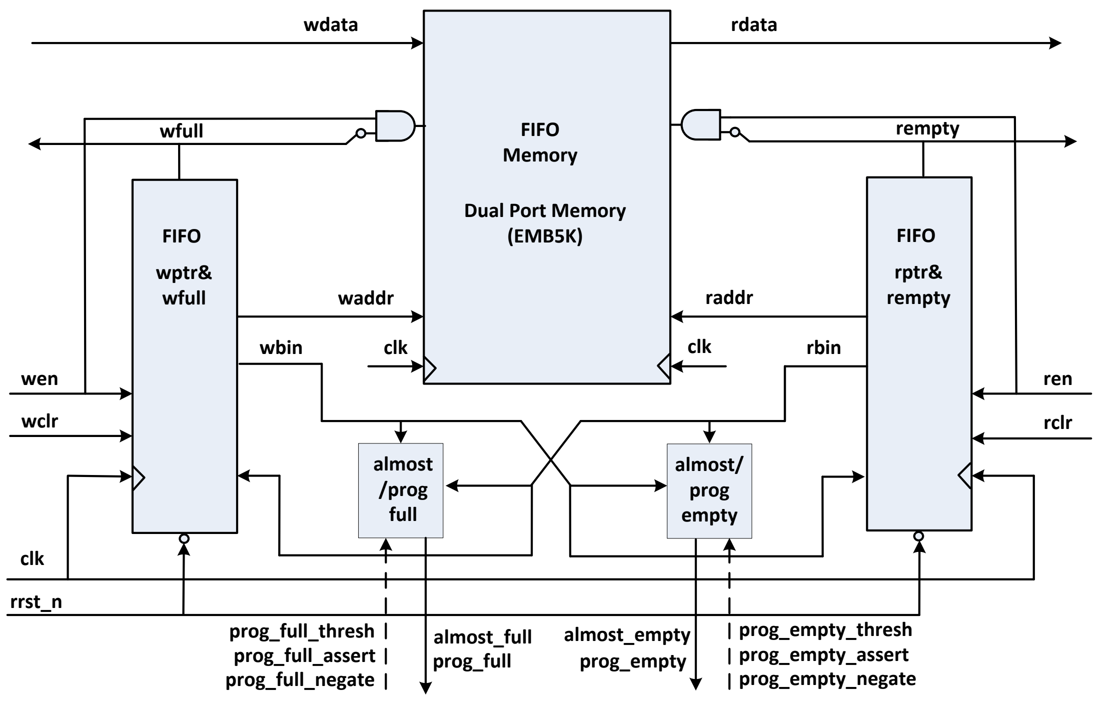

*Figure 2-2* ***Synchronous FIFO Block Diagram***

To implement synchronous FIFO function:

Compared with asynchronous FIFO, no synchronous bridge and gray code counter are needed in synchronous FIFO.

# 3 Software FIFO Interface

## 3.1 Asynchronous FIFO interface

Following figure shows the interface of asynchronous FIFO.

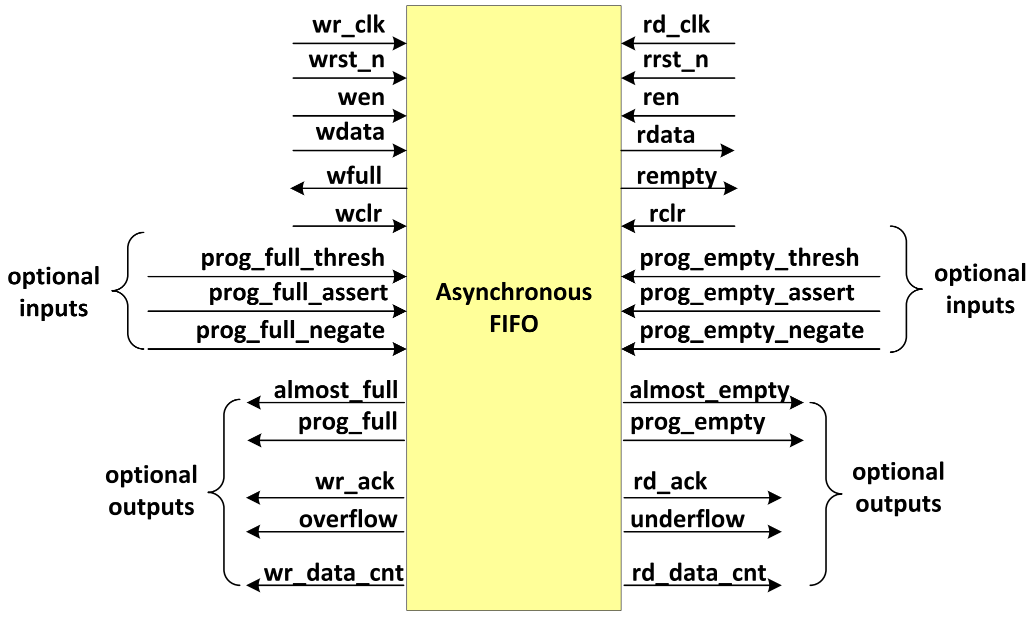

*Figure 3-1* ***Asynchronous FIFO Interface***

The asynchronous FIFO pin descriptions are outlined in the table below.

*Table 3-1* ***Asynchronous FIFO pin description***

| Interface | Name | Direction | Width | Description |
|---|---|---|---|---|
| System | wclk | Input | 1 | write clock input |
| | wrst_n | Input | 1 | Write reset input, active low |
| | rclk | Input | 1 | Read clock input |
| | rrst_n | Input | 1 | Read reset input, active low |
| User Interface | wdata | Input | wr_dw | FIFO write data |
| | rdata | Input | rd_dw | FIFO read data |
| | wen | Input | 1 | FIFO write enable |
| | ren | Input | 1 | FIFO read enable |
| | wclr | Input (optional) | 1 | Synchronous clear input to clear write pointer, write related flag output, active high |
| | rclr | Input (optional) | 1 | Synchronous clear input to clear read pointer, read related flag output, active high |
| | prog_full_thresh | Input (optional) | wr_aw | Programmable Full Threshold: This signal is used to input the |

| | | | | |
|---|---|---|---|---|
| | | | | threshold value for the assertion and de-assertion of the<br>programmable full flag. |
| | prog_empty_thresh | Input (optional) | rd_aw | Programmable Empty Threshold: This signal is used to input the<br>threshold value for the assertion and de-assertion of the<br>programmable empty flag. |
| | prog_full_assert | Input (optional) | wr_aw | Programmable Full Threshold Assert: This signal is used to set the upper threshold value for the programmable full flag, which defines when the signal is asserted. |
| | prog_full_negate | Input (optional) | wr_aw | Programmable Full Threshold Negate: This signal is used to set the<br>lower threshold value for the programmable full flag, which defines when the signal is de-asserted. |
| | prog_empty_assert | Input (optional) | rd_aw | Programmable Empty Threshold Assert: This signal is used to set<br>the lower threshold value for the programmable empty flag, which<br>defines when the signal is asserted |
| | prog_empty_negate | Input (optional) | rd_aw | Programmable Empty Threshold Negate: This signal is used to set<br>the upper threshold value for the programmable empty flag, which<br>defines when the signal is de-asserted. |
| | wfull | Output | 1 | FIFO full flag, active high |
| | rempty | Output | 1 | FIFO empty flag, active high |
| | almost_full | Output (optional) | 1 | This signal indicates that only one more write can be performed before the FIFO is full. |
| | almost_empty | Ouput (optional) | 1 | this signal indicates that the FIFO is almost empty and one word remains in the FIFO. |
| | prog_full | Output (optional) | 1 | This signal is asserted when the number of words in the FIFO is greater than or equal to the assert threshold. It is deasserted when the number of words in the FIFO is less than the threshold. |
| | prog_empty | Ouput (optional) | 1 | This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is de-asserted when the number of words in the FIFO exceeds the programmable threshold. |
| | wr_ack | Output (optional) | 1 | Write Acknowledge: This signal indicates that a write request (wen) during the prior clock cycle succeeded. |

| | | | | |
|---|---|---|---|---|
| | rd_ack | Output (optional) | 1 | rd_ack: This signal indicates that valid data is available on the output bus (rdata). |
| | overflow | Output (optional) | 1 | Overflow: This signal indicates that a write request (wen) during the prior clock cycle was rejected, because the FIFO is full. |
| | underflow | Output (optional) | 1 | Underflow: Indicates that read request (ren) during the previous clock cycle was rejected because the FIFO is empty. |
| | wr_data_cnt | Output (optional) | wr_aw | Indicate how many data are stored in FIFO, in write clock domain. |
| | rd_data_cnt | Output (optional) | rd_aw | Indicate how many data are stored in FIFO, in read clock domain. |

## 3.2 Synchronous FIFO interface

Following figure shows the synchronous FIFO interface

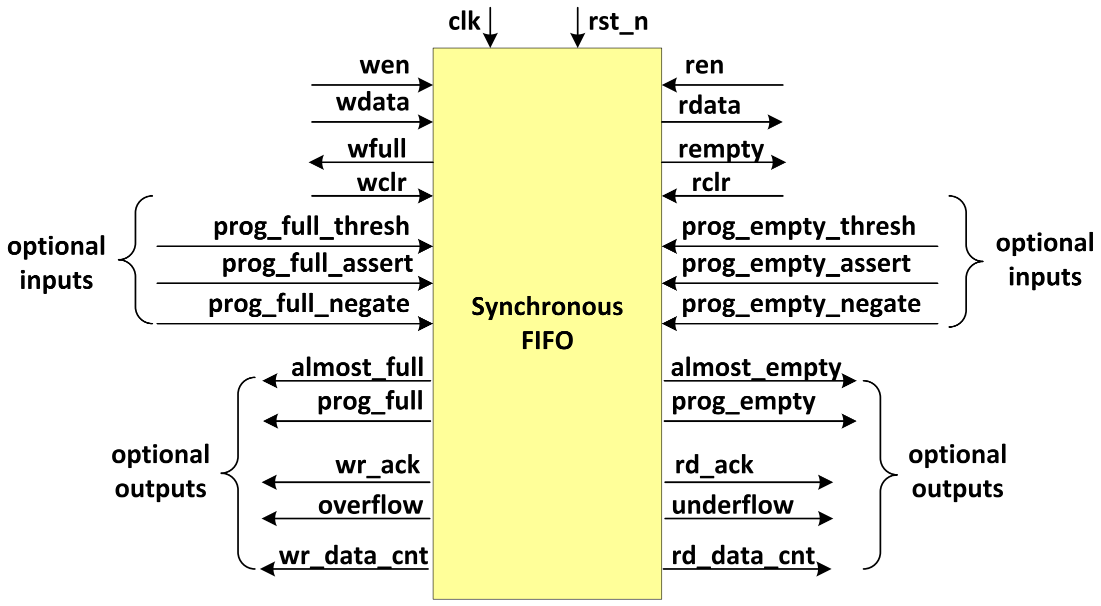

*Figure 3-2* ***Synchronous FIFO Interface***

The synchronous FIFO pin descriptions are outlined in the table below.

*Table 3-2* ***Synchronous FIFO pin description***

| Interface | Name | Direction | Width | Description |
|---|---|---|---|---|
| System | clk | Input | 1 | clock input |
| | rst_n | Input | 1 | reset input, active low |

| | | | | |
|---|---|---|---|---|
| User Interface | wdata | Input | wr_dw | FIFO write data |
| | rdata | Input | rd_dw | FIFO read data |
| | wen | Input | 1 | FIFO write enable |
| | ren | Input | 1 | FIFO read enable |
| | wclr | Input (optional) | 1 | Synchronous clear input to clear write pointer, write related flag output, active high |
| | rclr | Input (optional) | 1 | Synchronous clear input to clear read pointer, read related flag output, active high |
| | prog_full_thresh | Input (optional) | wr_aw | Programmable Full Threshold: This signal is used to input the<br>threshold value for the assertion and de-assertion of the<br>programmable full flag. |
| | prog_empty_thresh | Input (optional) | rd_aw | Programmable Empty Threshold: This signal is used to input the<br>threshold value for the assertion and de-assertion of the<br>programmable empty flag. |
| | prog_full_assert | Input (optional) | wr_aw | Programmable Full Threshold Assert: This signal is used to set the upper threshold value for the programmable full flag, which defines when the signal is asserted. |
| | prog_full_negate | Input (optional) | wr_aw | Programmable Full Threshold Negate: This signal is used to set the<br>lower threshold value for the programmable full flag, which defines when the signal is de-asserted. |
| | prog_empty_assert | Input (optional) | rd_aw | Programmable Empty Threshold Assert: This signal is used to set<br>the lower threshold value for the programmable empty flag, which<br>defines when the signal is asserted |
| | prog_empty_negate | Input (optional) | rd_aw | Programmable Empty Threshold Negate: This signal is used to set<br>the upper threshold value for the programmable empty flag, which<br>defines when the signal is de-asserted. |
| | wfull | Output | 1 | FIFO full flag, active high |
| | rempty | Output | 1 | FIFO empty flag, active high |
| | almost_full | Output (optional) | 1 | This signal indicates that only one more write can be performed before the FIFO is full. |
| | almost_empty | Ouput (optional) | 1 | This signal indicates that the FIFO is almost empty and one word remains in the FIFO. |
| | prog_full | Output (optional) | 1 | This signal is asserted when the number of words in the FIFO is greater than or equal to the assert |

| | | | | |
|---|---|---|---|---|
| | | | | threshold. It is deasserted when the number of words in the FIFO is less than the negate threshold. The threshold can be const, dynamic inputs, const range or dynamic range inputs |
| | prog_empty | Ouput (optional) | 1 | This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is de-asserted when the number of words in the FIFO exceeds the programmable threshold. The threshold can be const, dynamic inputs, const range or dynamic range inputs |
| | rd_ack | Output (optional) | 1 | Rd_ack: This signal indicates that valid data is available on the output bus (rdata). |
| | wr_ack | Output (optional) | 1 | Write Acknowledge: This signal indicates that a write request (wen) during the prior clock cycle succeeded. |
| | overflow | Output (optional) | 1 | Overflow: This signal indicates that a write request (wen) during the prior clock cycle was rejected, because the FIFO is full. |
| | underflow | Output (optional) | 1 | Underflow: Indicates that read request (ren) during the previous clock cycle was rejected because the FIFO is empty. |
| | wr_data_cnt | Output (optional) | wr_aw | Indicate how many data are stored in FIFO in write domain |
| | rd_data_cnt | Output (optional) | rd_aw | Indicate how many data are stored in FIFO in read domain |

## 3.3 FIFO programmed parameter list

FIFO related parameters are shown as below:

*Table 3-3* ***FIFO parameters list***

| Name | Type | Value | Description |
|---|---|---|---|
| wr_dw | int | >=1 | Write data width |
| rd_dw | int | >=1 | Read data width |
| wr_aw | int | >=4 | Write address width |
| rd_aw | Int | >=4 | Read address width |
| need_syn_wr_clr | Int | 1, 0 | 1: Synchronous clear for FIFO in write clock domain;<br>0: Not synchronous clear |
| need_syn_rd_clr | int | 1, 0 | 1: Synchronous clear for FIFO in write clock domain;<br>0: Not synchronous clear |

| | | | |
|---|---|---|---|
| prog_full_type | string | const, dyn_single range, dyn_range | Support 4 types of programmable full flag generation:<br>const: single programmable threshold setting through parameter;<br>dyn_single: single programmable threshold setting through inputs(prog_full_thresh);<br>range: multiple programmable threshold setting through parameters;<br>dyn_range: multiple programmable thresh setting through inputs ports |
| prog_empty_type | String | const, dyn_single range, dyn_range | Support 4 types ofprogrammable full flag generation:<br>const: single programmable threshold setting through parameter;<br>dyn_single:<br>single programmable threshold setting through inputs(prog_full_thresh);<br>range:<br>multiple programmable threshold setting through parameters;<br>dyn_range:<br>multiple programmable thresh setting through inputs ports |
| prog_full_thresh | Int | >=2,<=wn, wn=(1<<wr_aw -2*sh_bits) | Programmable full threshold |
| prog_empty_thresh | Int | >=2, <=rn rn=(1<<rd_aw-2*sh_bits ) | Programmable empty threshold |
| prog_full_assert_const | Int | >=3,<=wn, wn=(1<<wr_aw -2*sh_bits) | Programmable upper threshold value for the programmable full flag |
| prog_full_negate_const | int | >=2,<=wn, wn=(1<<wr_aw -3*sh_bits) | Programmable lower threshold value for the programmable full flag, must less than prog_full_assert_const |
| prog_empty_assert_const | Int | >=2, <=rn rn=(1<<rd_aw-3*sh_bits ) | Programmable lower threshold value for the programmable empty flag |

| | | | |
|---|---|---|---|
| prog_empty_negate_const | Int | >=3, <=rn<br>rn=(1<<rd_aw-2*sh_bits<br>) | Programmable upper threshold value for the programmable empty flag, must bigger than prog_empty_asser_const |
| gen_prog_full | Int | 1, 0 | 1: generate programmable full flag；<br>0: do not generate programmable full flag. |
| gen_prog_empty | Int | 1, 0 | 1: generate programmable empty flag；<br>0: do not generate programmable empty flag. |
| gen_almost_full | Int | 1, 0 | 1: generate almost full flag；<br>0: do not generate almost full flag. |
| gen_almost_empty | Int | 1, 0 | 1: generate almost empty flag；<br>0: do not generate almost empty flag. |
| gen_wr_data_cnt | Int | 1, 0 | 1: generate write data count flag；<br>0: do not generate write data count flag. |
| gen_rd_data_cnt | Int | 1, 0 | 1: generate read data count flag；<br>0: do not generate read data count flag. |
| gen_wr_ack | Int | 1, 0 | 1: generate write acknowledge flag；<br>0: do not generate write acknowledge flag. |
| gen_rd_ack | Int | 1, 0 | 1: generate read acknowledge flag；<br>0: do not generate read acknowledge flag. |
| gen_wr_overflow | Int | 1, 0 | 1: generate write overflow flag;<br>0: do not generate write overflow flag. |
| gen_rd_underflow | int | 1, 0 | 1: generate read underflow flag;<br>0: do not generate read underflow flag. |
| fwft_en | int | 1,0 | 1: support FWFT function<br>0: not support FWFT function |

NOTE:

sh_bits = wr_dw > rd_dw ? log2(wr_dw/rd_dw) : log2(rd_dw/wr_dw)

# 4 Software FIFO Usage and Control

This section describes the behavior of asynchronous and synchronous FIFO write/read and the associated status flags.

When write enable is asserted and the FIFO is not full, data is added to the FIFO from the input bus and write acknowledge is asserted. If the FIFO is continuously written without being read, it is filled with data. Write operations are successful only when the FIFO is not full. When the FIFO is full, any writing request is ignored, in the meanwhile the overflow flag is asserted and there is no change in the state of the FIFO.

When read enable signal is asserted and the FIFO is not empty, data is read from the FIFO on the output bus, and the rd_ack flag is asserted. If the FIFO is continuously read without being written, the FIFO empties eventually. Read operations are successful only if the FIFO is not empty. When the FIFO is empty, any read operation request is ignored, meanwhile the underflow flag is asserted and there is no change in the state of the FIFO.

## 4.1 FIFO Full/Empty generation

A FIFO is full when the pointers are equal again, that is, the write pointer has wrapped around and caught up to the read pointer.

A FIFO is empty when the read and write pointers are both equal. This condition happens when both pointers are reset to zero during a reset operation, or when the read pointer catches up to the write pointer, which means the last data is read out from the FIFO.

**Full Generation:**

- **Asynchronous FIFO full generation example**

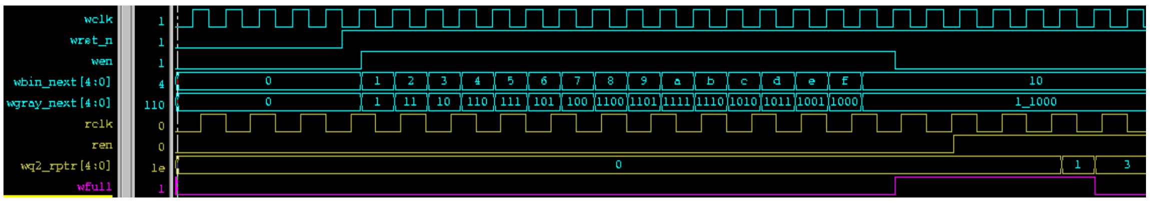

*Figure 4-1* ***Asynchronous FIFO full generation***

- **Synchronous FIFO full generation example**

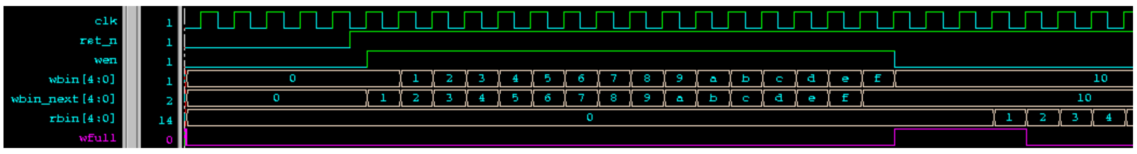

*Figure 4-2* ***Synchronous FIFO full generation***

**Empty Generation:**

- **Asynchronous FIFO empty generation example**

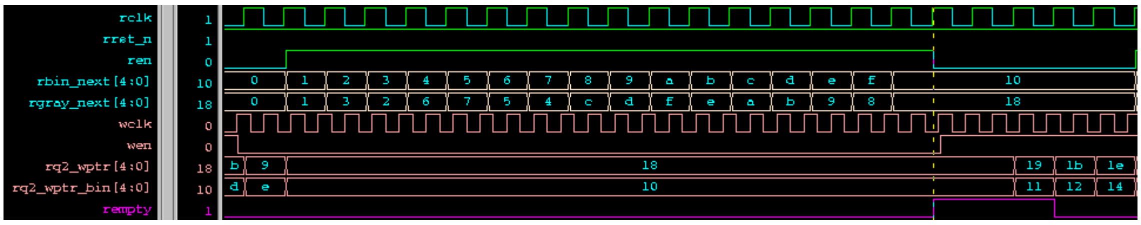

*Figure 4-3 A**synchronous FIFO empty generation***

- **Synchronous FIFO empty generation example**

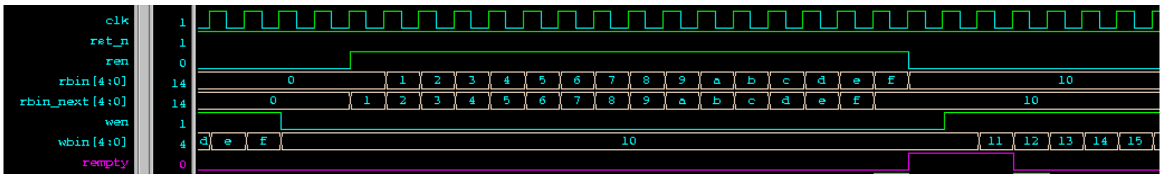

*Figure 4-4 **Synchronous FIFO empty generation***

## 4.2 Almost full/Almost empty generation

When only one more write operation can be performed before the FIFO is full, the almost full signal is asserted. This flag is active high and synchronous to the write clock.

When only one more read operation can be performed before the FIFO is empty, the almost empty signal is asserted. This flag is active high and synchronous to the read clock.

**Almost full generation:**

- **Asynchronous FIFO almost full generation example**

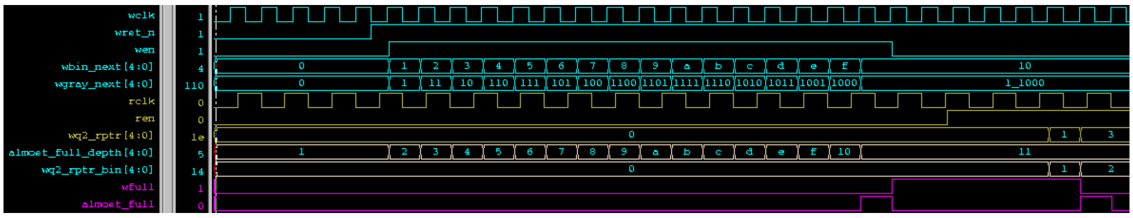

*Figure 4-5 **Asynchronous FIFO almost full generation***

- **Synchronous FIFO almost full generation example**

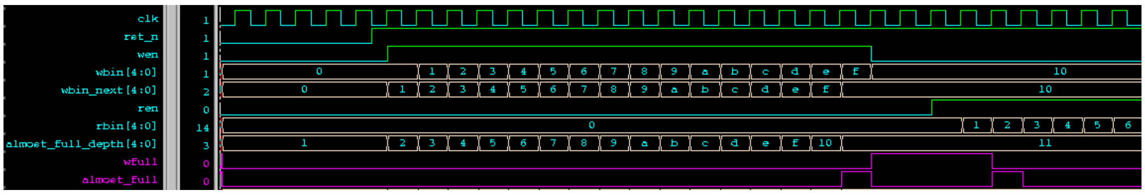

*Figure 4-6 **Synchronous FIFO almost full generation***

**Almost empty generation:**

- **Asynchronous FIFO almost empty generation example**

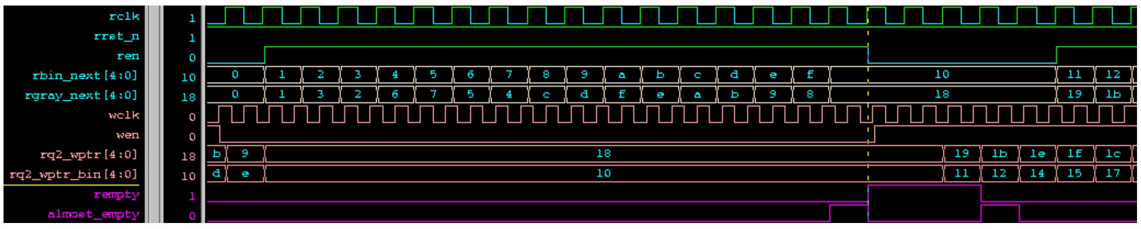

*Figure 4-7* ***Asynchronous FIFO almost empty generation***

- **Synchronous FIFO almost empty generation example**

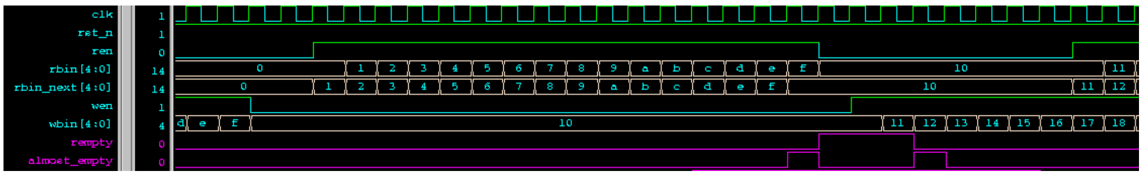

*Figure 4-8* ***Synchronous FIFO almost empty generation***

## 4.3 Programmable full/empty generation

### 4.3.1 Programmable Full

The FIFO Generator supports four ways to define the programmable full threshold:

- Single threshold constant (prog_full_type = “single”)
- Single threshold with dedicated input port (prog_full_typ = “dyn_single”)
- Assert and negate threshold constants (prog_full_type = “range”)
- Assert and negate thresholds with dedicated input ports (prog_full_type = “dyn_range”)

When the number of words in the FIFO is greater than or equal to the asserted threshold, the programmable full signal is asserted. It is de-asserted when the number of words in the FIFO less than the negate threshold. When the number of words in the FIFO is less than or equal to the programmable threshold, the programmable empty signal is asserted. It is de-asserted when the number of words in the FIFO exceeds the programmable threshold.

#### Programmable Full: Single Threshold

This option enables the user to set a single threshold value for the assertion and deassertion of prog_full. When the number of entries in the FIFO is greater than or equal to the threshold value, prog_full is asserted or else the flag is deasserted.

Two options are available to implement this threshold:

- Single threshold constant. User specifies the threshold value through the IP Wizard. Once the core is generated, this value can only be changed by regenerating the core. This option consumes fewer resources than the single threshold with dedicated input port.
- Single threshold with dedicated input port. User specifies the threshold value through an input port (prog_full_thresh) on the core. This input can be changed dynamically, providing the user the flexibility to change the programmable full threshold in-circuit without re-generating the core.

- **Asynchronous FIFO programmable full generation example**

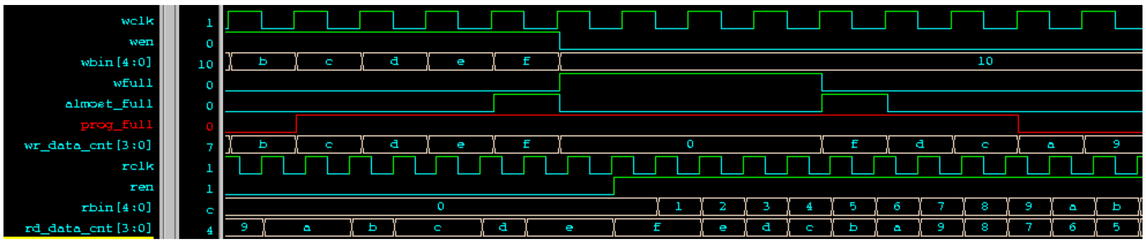

*Figure 4-9* ***Asynchronous FIFO programmable full generation (prog_full_thresh=12)***

- **Synchronous FIFO programmable full generation example**

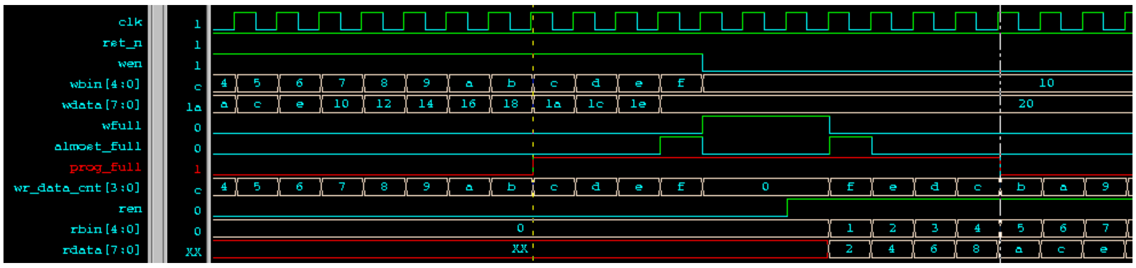

*Figure 4-10* ***Synchronous FIFO programmable full generation (prog_full_thresh=12)***

## Programmable Full: Assert and Negate Thresholds

This option enables the user to set separate values for the assertion and deassertion of prog_full. When the number of entries in the FIFO is greater than or equal to the assert value, prog_full is asserted. When the number of entries in the FIFO is less than the negate value, prog_full is deasserted.

Two options are available to implement these thresholds:

- **Assert and negate threshold constants:** User specifies the threshold values through the IP Wizard. Once the core is generated, these values can only be changed by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.
- **Assert and negate thresholds with dedicated input ports:** User specifies the threshold values through input ports on the core. These input ports can be changed dynamically, providing the user with the flexibility to change the values of the programmable full assert (prog_full_assert) and negate (prog_full_negate) thresholds in-circuit without re-generating the core.

- **Asynchronous FIFO programmable full generation example**

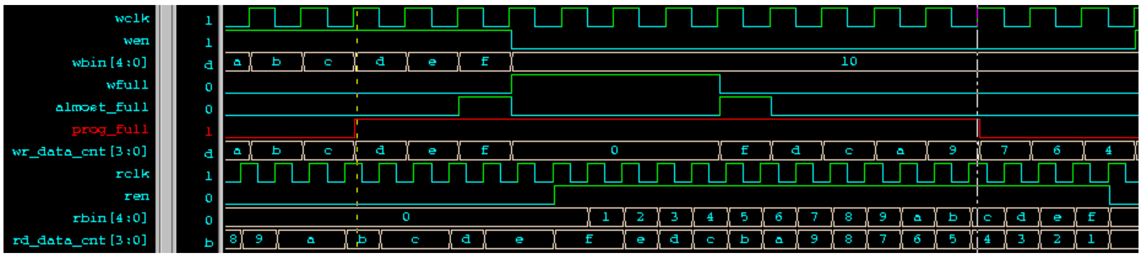

*Figure 4-11* ***Asynchronous FIFO programmable full generation (prog_full_assert=13, prog_full_negate=9)***

- **Synchronous FIFO programmable full generation example**

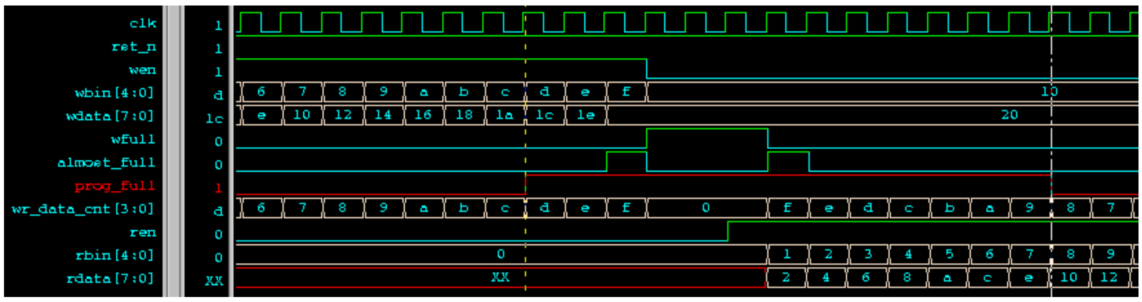

*Figure 4-12* ***Synchronous FIFO programmable full generation***

***(prog_full_assert=13, prog_full_negate=9)***

### 4.3.2 Programmable Empty

The FIFO Generator supports four ways to define the programmable empty thresholds:

- Single threshold constant (prog_empty_type = “single”)
- Single threshold with dedicated input port (prog_empty_type = “dyn_single”)
- Assert and negate threshold constants (prog_empty_type = “range”)
- Assert and negate thresholds with dedicated input ports (prog_empty_type = “dyn_range”)

#### Programmable Empty: Single Threshold

This option enables you to set a single threshold value for the assertion and deassertion of prog_empty. When the number of entries in the FIFO is less than or equal to the threshold value, prog_empty is asserted or else prog_empty is deasserted.

Two options are available to implement this threshold:

- **Single threshold constant:** User specifies the threshold value through the IP Wizard. Once the core is generated, this value can only be changed by regenerating the core. This option consumes fewer resources than the single threshold with dedicated input port.
- **Single threshold with dedicated input port**: User specifies the threshold value through an input port (prog_empty_thresh) on the core. This input can be changed dynamically, providing the flexibility to change the programmable empty threshold in-circuit without re-generating the core.

■ **Asynchronous FIFO programmable empty generation example**

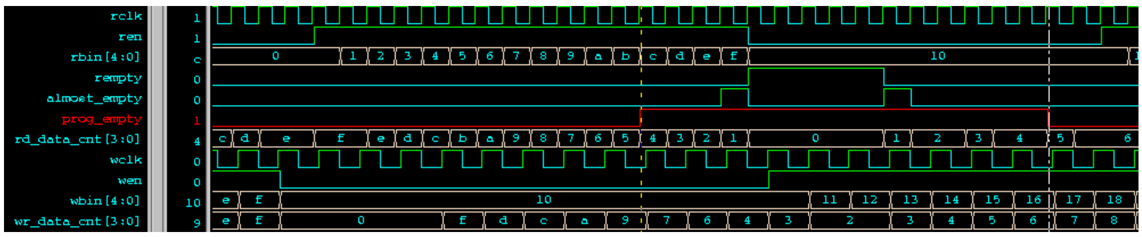

*Figure 4-13* ***Asynchronous FIFO programmable empty generation (prog_empty_thresh=4)***

■ **Synchronous FIFO programmable empty generation example**

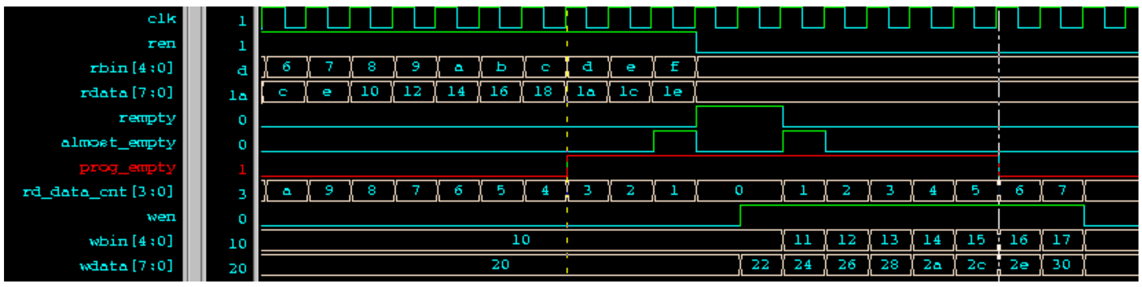

*Figure 4-14* ***Synchronous FIFO programmable empty generation (prog_empty_thresh=4)***

**Programmable Empty: Assert and Negate Thresholds**

This option lets the user set separate values for the assertion and deassertion of prog_empty. When the number of entries in the FIFO is less than or equal to the assert value, prog_empty is asserted. When the number of entries in the FIFO is greater than the negate value, prog_empty is deasserted.

Two options are available to implement these thresholds.

- **Assert and negate threshold constants**. The threshold values are specified through the IP Wizard. Once the core is generated, these values can only be changed by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.
- **Assert and negate thresholds with dedicated input ports**. The threshold values are specified through input ports on the core. These input ports can be changed dynamically, providing the user the flexibility to change the values of the programmable empty assert (prog_empty_assert) and negate (prog_empty_negate) thresholds in-circuit without regenerating the core.

■ **Asynchronous FIFO programmable empty generation example**

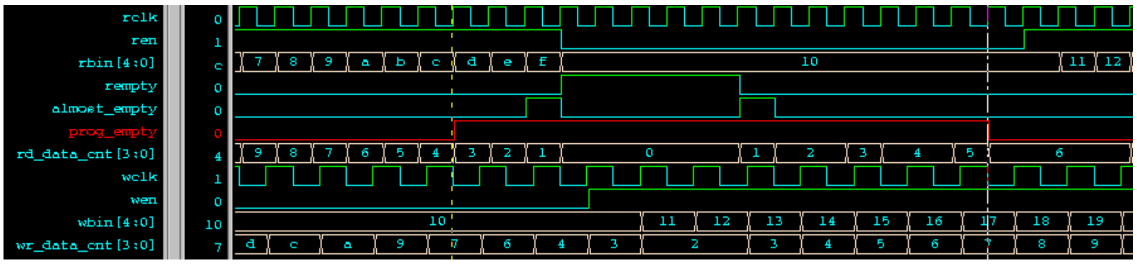

*Figure 4-15* ***Asynchronous FIFO programmable empty generation (prog_empty_assert=3, prog_empty_negate=5)***

■ **Synchronous FIFO programmable empty generation example**

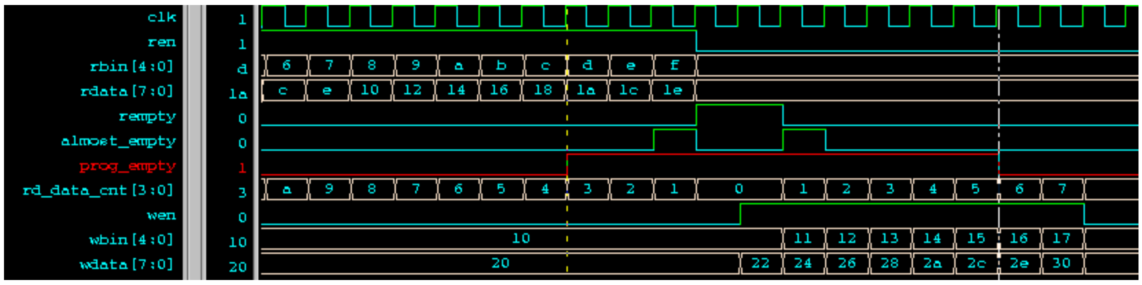

*Figure 4-16* ***Synchronous FIFO programmable empty generation (prog_empty_assert=3, prog_empty_negate=5)***

### 4.3.3 Data count generation

Write data count (wr_data_cnt) pessimistically reports the number of words written into the FIFO. The count is guaranteed to never under-report the number of words in the FIFO (although it may temporarily over-report the number of words present) to ensure that the user never overflows the FIFO.
Read data count (rd_data_cnt) pessimistically reports the number of words available for reading. The count is guaranteed to never over-report the number of words available in the FIFO (although it may temporarily under-report the number of words available) to ensure that the user design never underflows the FIFO.
For write and read with different width, the wr_data_cnt represents the data with write data width that stored in FIFO and rd_data_cnt represents the data with read data width that stored in FIFO.

■ **Asynchronous FIFO**

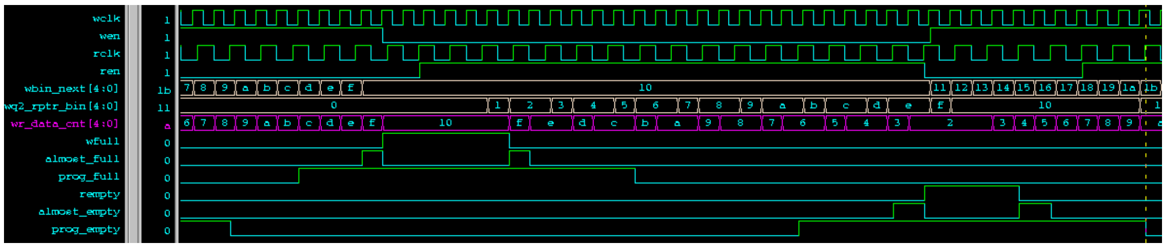

*Figure4-17* ***Asynchronous FIFO wr_data_cnt (prog_empty_thresh=4)***

The wr_data_cnt signal shows how many data are stored in FIFO in write clock domain, as is shown in the figure above

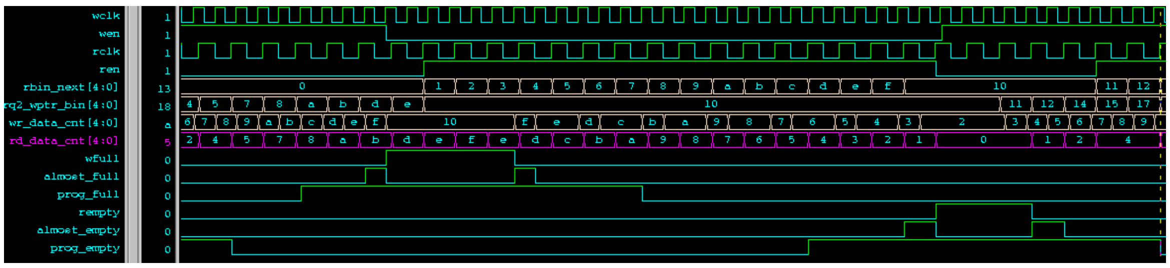

*Figure4-18* ***Asynchronous FIFO rd_data_cnt (prog_empty_thresh=4)***

The rd_data_cnt signal shows how many data are stored in FIFO in read clock domain, as is shown in the figure above

■ **Synchronous FIFO**

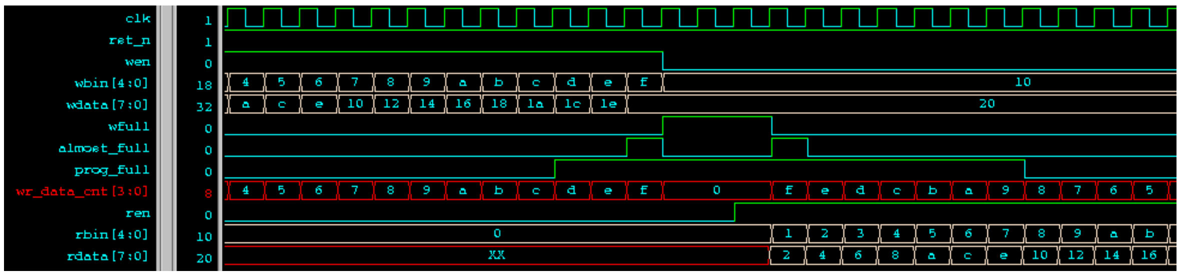

*Figure4-19* ***Synchronous FIFO wr_data_cnt (prog_full_assert=13, prog_full_negate=9)***

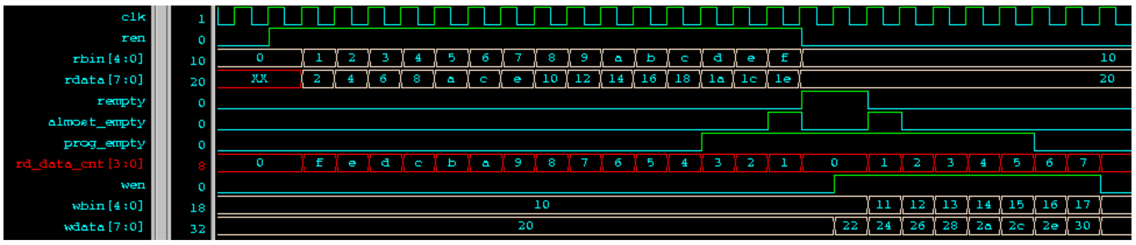

*Figure4-20* ***Synchronous FIFO rd_data_cnt (prog_full_assert=13, prog_full_negate=9)***

### 4.3.4 Handshaking interface generation

Handshaking flags (read acknowledge, underflow, write acknowledge and overflow) are supported to provide additional information regarding the status of the write and read operations.

The write acknowledge flag (wr_ack) is asserted at the completion of each successful write operation and indicates that the data on the wdata port has been stored in the FIFO. This flag is synchronous to the write clock (wclk).

The read acknowledge flag (rd_ack) is asserted at the completion of each successful read operation and indicates that the data read out from FIFO. This flag is synchronous to the read clock (rclk).

The over_flow flag, this signal indicates that a write request (wen) during the prior clock cycle is rejected, because the FIFO is full

The underflow flag, this signal indicates that the read request during the previous clock cycle was rejected because the FIFO is empty

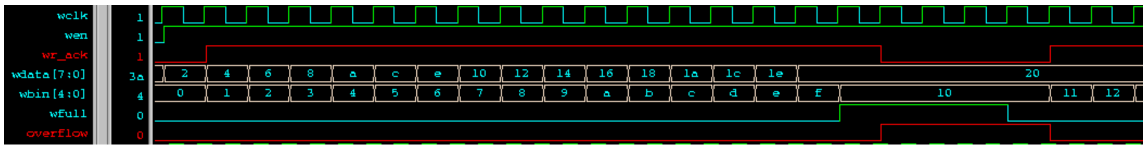

*Figure4-21* ***Asynchronous FIFO write handshaking signal***

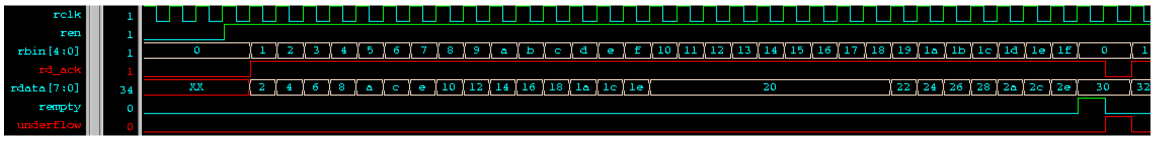

*Figure4-22* ***Asynchronous FIFO read handshaking signal***

For synchronous FIFO, the handshaking interface is the same as asynchronous FIFO, except that synchronous FIFO's write and read are in the same clock domain.

# 5 Hardware FIFO

## 5.1 Introduction

The hardware FIFO can support the features as below:

- Support data width up to 18 bits with depth up to 1K bits;
- Support both synchronous and asynchronous FIFO;
- Support wfull/rempty status flag;
- Optional synchronous clear inputs that can clear write and read pointer simultaneously ;
- Optional almost_full/almost_empty status flag generated according to the single threshold constant.
- Four optional handshake signals (wr_ack, rd_ack, overflow, underflow) provide feedback ( acknowledgment or rejection) in response to write and read requests in the prior clock cycle;
- Invalid read or write requests are rejected without affecting the FIFO state. When FIFO is full, data can't write to FIFO anymore, the write data will be omitted. When FIFO is empty, data can't read out anymore;

Device Support:

HME-M7

## 5.2 Difference between SW FIFO and HW FIFO

*Table 5-1* ***Compare SW FIFO with HW FIFO***

| **Items** / **FIFO** | | **SW FIFO** | **HW FIFO** |
|---|---|---|---|
| Size | | Depth up to: $2^{19}$<br>Width up to: 288 | Only support up to 18Kbits |
| Pins | almost full/empty | Indicates there is only one more data can be written to or read from FIFO | Same as prog_full/empty in SW FIFO |
| | wr_cnt<br>rd_cnt | Indicates the number of data stored in FIFO in write/read clock domain | Do not support this function |
| | clear signal | wclr: clear write pointer<br>rclr: clear read pointer | fifo_clr: clear write and read pointer simultaneously |
| Features | Proggrammble full/empty threshold | Support 4 types:<br>Single threshold constant, Single threshold with dedicated input port, | Only one type:<br>Single threshold constant |

| | | | |
|---|---|---|---|
| | | Assert and negate threshold constants,<br>Assert and negate thresholds with dedicated input ports | |

# 6 AHB interface FIFO

See the document of "HME_AHB_interface_FIFO_user_guide_EN02.pdf" please click the file name.

# 7 Resource usage

*Table 7-1* ***Benchmarks, synchronous FIFO without optional features***

| DepthxWidth | LUTs | Regs | EMBs |
|---|---|---|---|
| 64x16 | 24 | 16 | 1 |
| 512x16 | 32 | 22 | 2 |
| 1024x16 | 52 | 25 | 4 |
| 2048x16 | 55 | 27 | 8 |
| 4096x16 | 57 | 29 | 16 |

*Table 7-2* ***Benchmarks, synchronous FIFO with almost full/empty flag, multiple threshold and handshaking***

| DepthxWidth | LUTs | Regs | EMBs |
|---|---|---|---|
| 64x16 | 46 | 22 | 1 |
| 512x16 | 60 | 28 | 2 |
| 1024x16 | 81 | 31 | 4 |
| 2048x16 | 87 | 33 | 8 |
| 4096x16 | 91 | 35 | 16 |

*Table 7-3* ***Benchmarks, asynchronous FIFO without optional features***

| DepthxWidth | LUTs | Regs | EMBs |
|---|---|---|---|
| 64x16 | 49 | 58 | 1 |
| 512x16 | 69 | 82 | 2 |
| 1024x16 | 93 | 91 | 4 |
| 2048x16 | 101 | 99 | 8 |
| 4096x16 | 110 | 107 | 16 |

*Table 7-4* ***Benchmarks, asynchronous FIFO with almost full/empty flag, multiple threshold and handshaking***

| DepthxWidth | LUTs | Regs | EMBs |
|---|---|---|---|
| 64x16 | 79 | 64 | 1 |
| 512x16 | 113 | 88 | 2 |
| 1024x16 | 142 | 97 | 4 |
| 2048x16 | 161 | 105 | 8 |
| 4096x16 | 169 | 113 | 16 |

# 8 Simulating your design

The Fuxi IP Wizard provide the source RTL for the FIFO, user can directly use this source code to do simulation. Except the RTL, the simulation libraries for HME-M5 and HME_M7 are needed, you can find the lib from:
Fuxi install directory\data\lib\js_sim.v
Fuxi install directory\data\lib\m7s_sim.v
Fuxi install directory\data\lib\hr3_sim.v

When using Fuxi IP Wizard to generate FIFO core, a simulation directory under ip_core\fifo_v2 is generated. You can find the source RTL under the ip_core\fifo\src directory, as is shown in Table9-1.

# 9 Generated File Directory Structure

The FIFO IP wizard generated code includes source files (src) and related documentation(doc). The detailed design directory structure is as below：

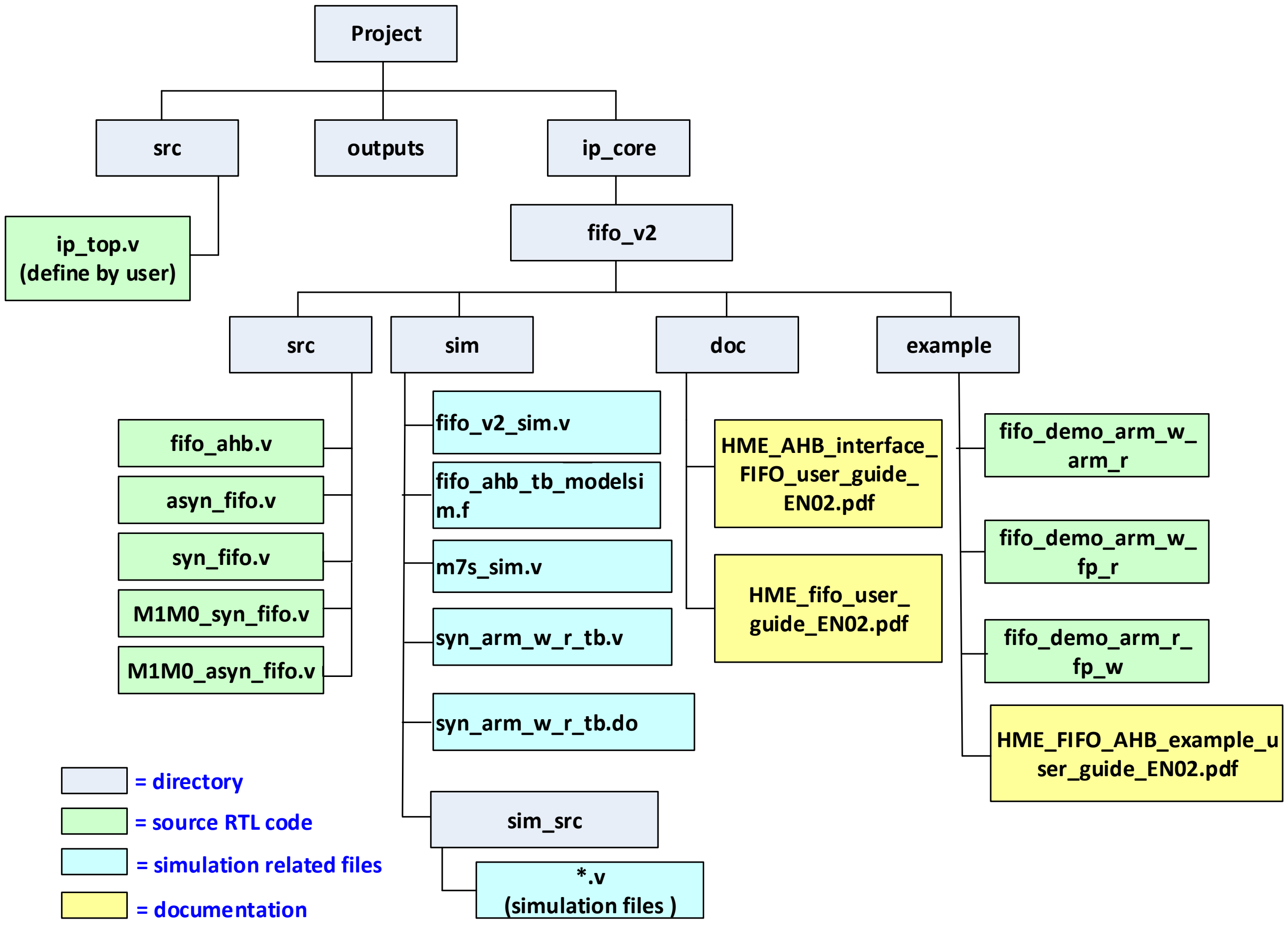

*Figure 9-1* ***FIFO IP wizard generate files structure***

*Table 9-1* ***Design Directory structure***

| **Directory** | **Description** |
|---|---|
| src\ | Directory for project source code, including IP wizard generate code. |
| ip_core\ | The directory specially for all IPs |
| \fifo_v2 | Directory for FIFO IP |
| \doc\HME_AHB_interface_FIFO_user_guide_EN02.doc | User guide for ahb interface FIFO IP |
| \doc\HME_fifo_user_guide_EN02.doc | User guide for FIFO IP (no ahb interface) |

| | |
|---|---|
| \src | IP Design RTL |
| fifo_ahb.v | The src of ahb interface FIFO IP |
| asyn_fifo.v | Asynchronous FIFO source code |
| syn_fifo.v | Synchronous FIFO source code |
| M1M0_syn_fifo.v | Synchronous FIFO source code for device M0/M1 |
| M1M0_asyn_fifo.v | Asynchronous FIFO source code for device M0/M1 |
| \sim | |
| \ syn_arm_w_r_tb.v | Testbench of ahb interface FIFO(arm write & arm read) IP |
| \ syn_arm_w_r_tb.do | Do script for Modelsim simulation (arm write & arm read) |
| \fifo_ahb_tb_modelsim.f | Modelsim simulation related files |
| \fifo_v2_sim.v | Other RTL design files with AHB interface for simulation |
| \sim_src | |
| *.v | Other RTL design files for simulation about AHB bus |
| \example | |
| fifo_demo_arm_r_fp_w.zip | AHB interface FIFO IP examples |
| fifo_demo_arm_w_arm_r.zip | |
| fifo_demo_arm_w_fp_r.zip | |
| HME_FIFO_AHB_example_user_guide_EN02.pdf | The guide of AHB interface FIFO |

# Revision History

| Revision | Date | Comments |
|---|---|---|
| 1.0 | 2018-03-26 | Initial release |
| 1.0 | 2018-03-26 | Interface update，support more prog_full/empty generation, support handshaking interfaces<br>Performance and resource usage update |
| 1.1 | 2018-03-26 | Change parameter type of prog_full/empty_type from string to int<br>Fix a bug of almost_empty generation of syn&asyn fifo |
| 1.1 | 2018-03-26 | Add AHB interface |
| 2.0 | 2018-03-26 | Support First Word Fall Through function |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |