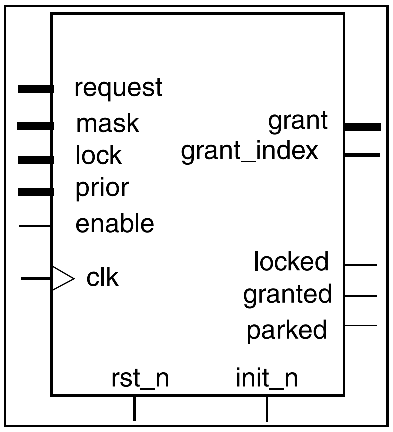
DW_arb_2t

Two-Tier Arbiter with Dynamic/Fair-Among-Equal Scheme

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterizable number of clients
- Programmable mask for all clients
- Park feature - default grant when no requests are pending
- Lock feature - ability to lock the currently granted client
- Registered/unregistered outputs



Applications

- Control application
- Networking
- Bus interfaces

Description

DW_arb_2t implements a parameterized, synchronous arbiter with a two-tiered arbitration scheme. In this scheme, each of the *n* clients of the arbiter can be programmed with a *p_width* bit `prior` input.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Asynchronous reset for all registers (active low)
init_n	1 bit	Input	Synchronous reset for all registers (active low)
enable	1 bit	Input	Enables clocking (active high)
request	*n* bit(s)	Input	Input request from clients
prior	*n*×*p_width* bit(s)	Input	Priority vector from the clients of the arbiter
lock	*n* bit(s)	Input	Active high signal to lock the grant to the current request. By setting `lock(i)` = 1, the arbiter is locked to the `request(i)` if it is currently granted. For `lock (i)` = 0, the lock on the arbiter is removed.

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
mask	*n* bit(s)	Input	Active high input to mask specific clients. By setting `mask(i)` = 1, `request(i)` is masked. For `mask(i)` = 0, the mask on the `request(i)` is removed.
parked	1 bit	Output	Flag to indicate that there are no requesting clients and the grant of resources has defaulted to *park_index*
granted	1 bit	Output	Flag to indicate that arbiter has issued a grant to one of the clients
locked	1 bit	Output	Flags that the arbiter is locked by a client
grant	*n* bit(s)	Output	Grant output
grant_index	ceil($\log_2 n$) bit(s)	Output	Index of the requesting client that has been currently granted or the client designated by *park_index* in *park_mode*

Table 1-2 Parameter Description

Parameter	Values	Description
n	2 to 32 Default: 4	Number of arbiter clients
p_width	1 to 5 Default: 2	Width of the priority vector of each client
park_mode	0 or 1 Default: 1	*park_mode* = 1 includes logic to enable parking when no clients are requesting and *park_mode* = 0 contains no logic for parking.
park_index	0 to *n*–1 Default: 0	Index of the client used for parking
output_mode	0 or 1 Default: 1	*output_mode* = 1 includes registers at the outputs (See Figure 1-2) *output_mode* = 0 contains no output registers (See Figure 1-3)

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis Model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW05.DW_ARB_2t_SIM_CFG	Design unit name for VHDL simulation
dw/dw05/DW_arb_2t_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_arb_2t.v	Verilog simulation model source code

Table 1-5 Arbiter Status Flags

Flag	Characteristic	Description
parked	If parked is active, there are no active requests at the input of the arbiter.	The parked output, active HIGH, indicates that grant of the resources has defaulted to the client defined by *park_index* in *park_mode* = 1. In *park_mode* = 0, this flag does not exist.
granted	If granted is active, there is at least one active request at the input of the arbiter.	The `granted` output, active HIGH, indicates that the grant of resources is to one of the actively requesting inputs.
locked	If locked is active, the current grant and the corresponding lock signal must be active.	The `locked` output, active HIGH, indicates that the currently granted client has locked out all other clients.

Primarily, the first-tier arbitration that uses the priority inputs of the clients, decides which one of the requesting clients is issued the grant signal. But in cases where two or more clients of the arbiter have the same programmed priority value the second-tier arbitration, based on fair-among-equal scheme, is used. In this scheme the grant is issued fairly, on a cycle by cycle basis for one cycle each, among actively requesting clients with same highest priority level.

The `lock` feature enables a client, despite requests from other clients, to have an exclusive grant for the duration of the corresponding lock input. After a client receives the grant, it can lock out other clients from the arbitration process by setting the corresponding `lock` input.

The park feature allows the resources to be granted to a designated client defined by the *park_index* parameter when there are no active requests pending. The *park_mode* and *lock_mode* parameters enable/disable these features.

By setting the desired bits of the `mask` input, the corresponding clients can be masked off from consideration for arbitration. The mask on a client remains active until the corresponding `mask` input for the client is reset.

All the input requests from the arbiter clients are assumed to be synchronous to the arbiter clock signal `clk`.

The arbiter provides flags: `locked`, `granted` and `parked`, to indicate the status of the arbiter. Table 1-5 shows a detailed description of all the flags of the arbiter.

Figure 1-1 A Two-Tiered Arbitration Scheme

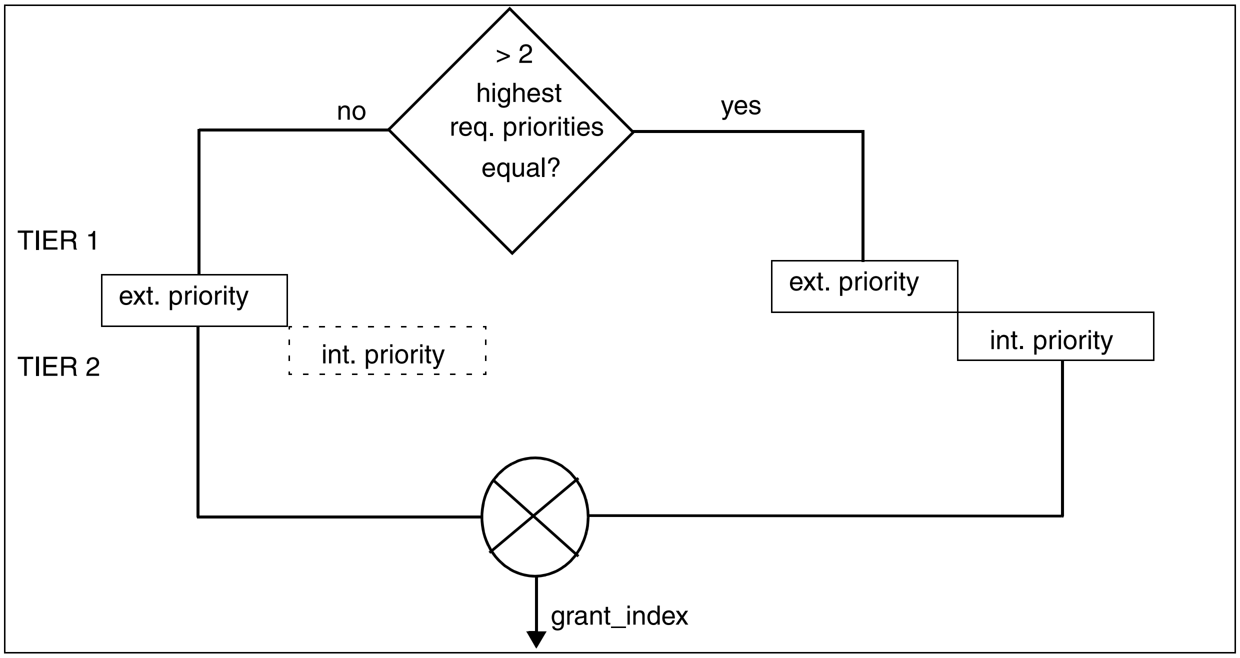


Figure 1-2 Block Diagram of DW_arb_2t Arbiter, *output_mode* = 1

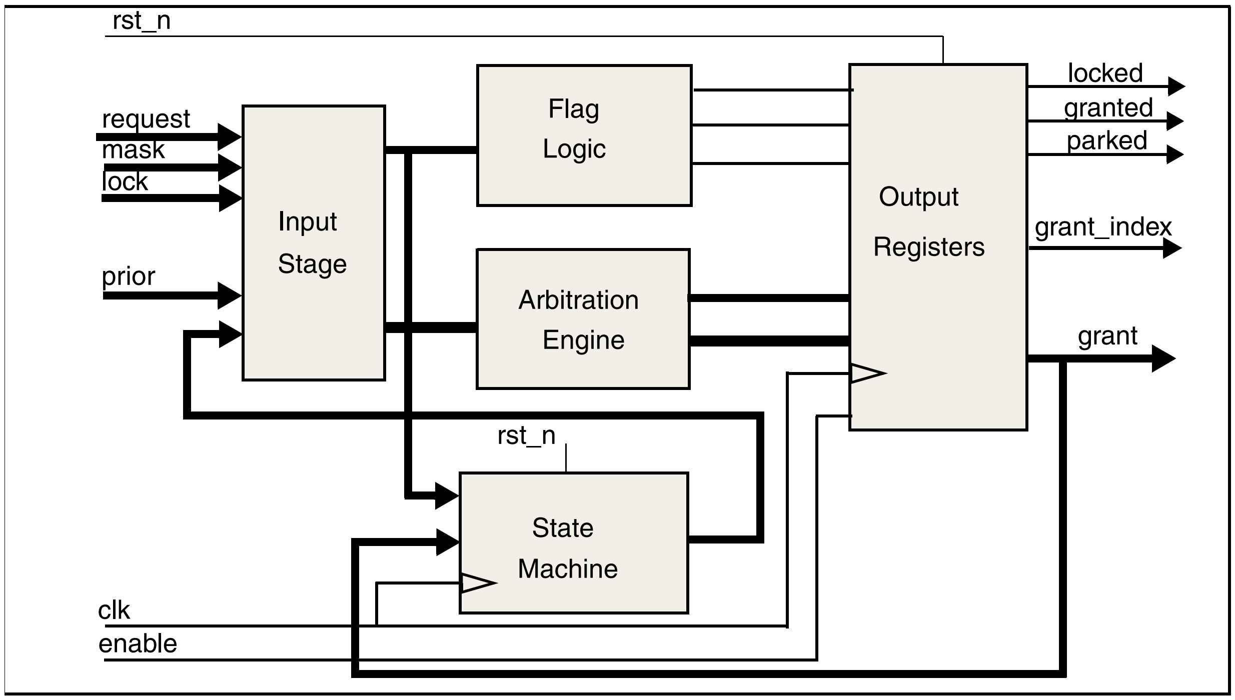
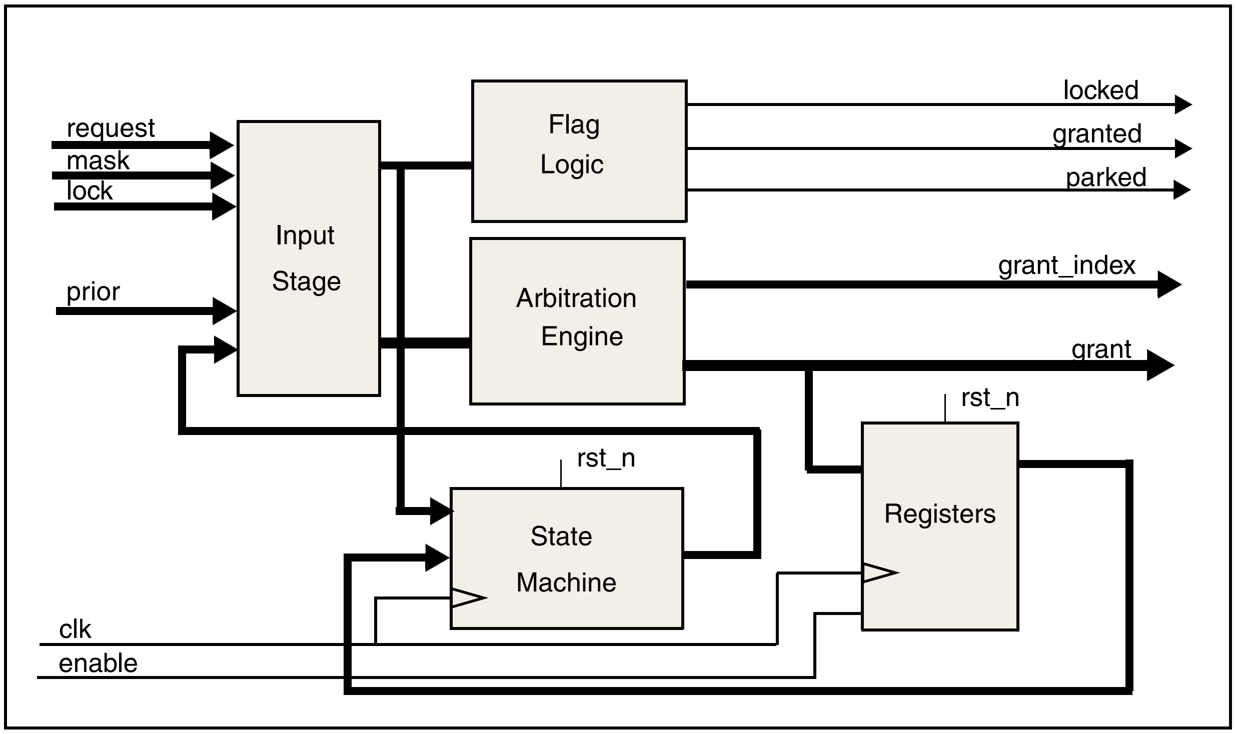


Figure 1-3 Block Diagram of DW_arb_2t Arbiter *output_mode* = 0



Functional Description

As stated earlier, the fair-among-equal arbitration scheme is one in which the `grant` is issued fairly, on a cycle by cycle basis for one cycle each, among the clients with same priority level as long as these clients are actively requesting. The `prior` input of the DW_arb_2t is a *n×p_width* wide vector formed by concatenation of the priority values of the *n* clients of the arbiter. See the examples in Figure 1-4, and Figure 1-5.

Figure 1-4 Priority Vector (prior)

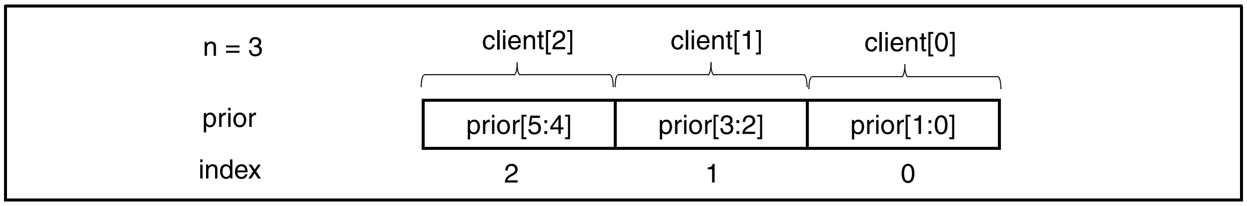
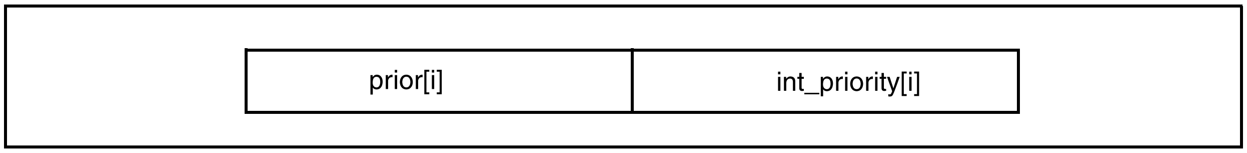


Figure 1-5 Concatenation of prior[i] and int_priority[i]



Each of the clients can be programmed with a priority level of 0 to $2^{**p\_width}-1$ and they need not be unique. In other words, two or more of the clients can be programmed with the same priority level. In such cases, if more than one of these clients are requesting and there are no other clients requesting at a higher priority level, the `grant` is issued to one among those clients on a fair-among-equal basis, for one cycle each.

The mask, park and lock features add flexibility to the arbiter. The parking of grant to a designated client saves an arbitration cycle and the parked client can lock the grant without issuing a request to the arbiter.

For the second tier arbitration, every input request of the DW_arb_2t maintains an internal register containing an **internal priority** (values from 0 to $2^{\mathrm{ceil}(\log_2 n)}-1$) that is updated each cycle. For each request, the internal priority is appended to the external input priority and the result is used by the arbiter to decide which of the requesting inputs gets the `grant`.

The internal priorities are updated based on the current state of the arbiter. The state diagram in Figure 1-1 on page 4 details the different states of the arbiter. The criteria used to update the priorities are as follows:

- The non-requesting inputs have their internal priorities set to lowest value of $2^{\mathrm{ceil}(\log_2 n)}-1$.
- The internal priorities of actively requesting inputs is increased by one each cycle until granted.
- The internal priority is rolled over to lowest priority value when highest priority is reached but the request is not granted. This happens when a client whose external priority is lower than currently granted and it has not been granted the resource in last *n* cycles.
- The internal priority of the currently granted client is set to the lowest value in the next cycle.

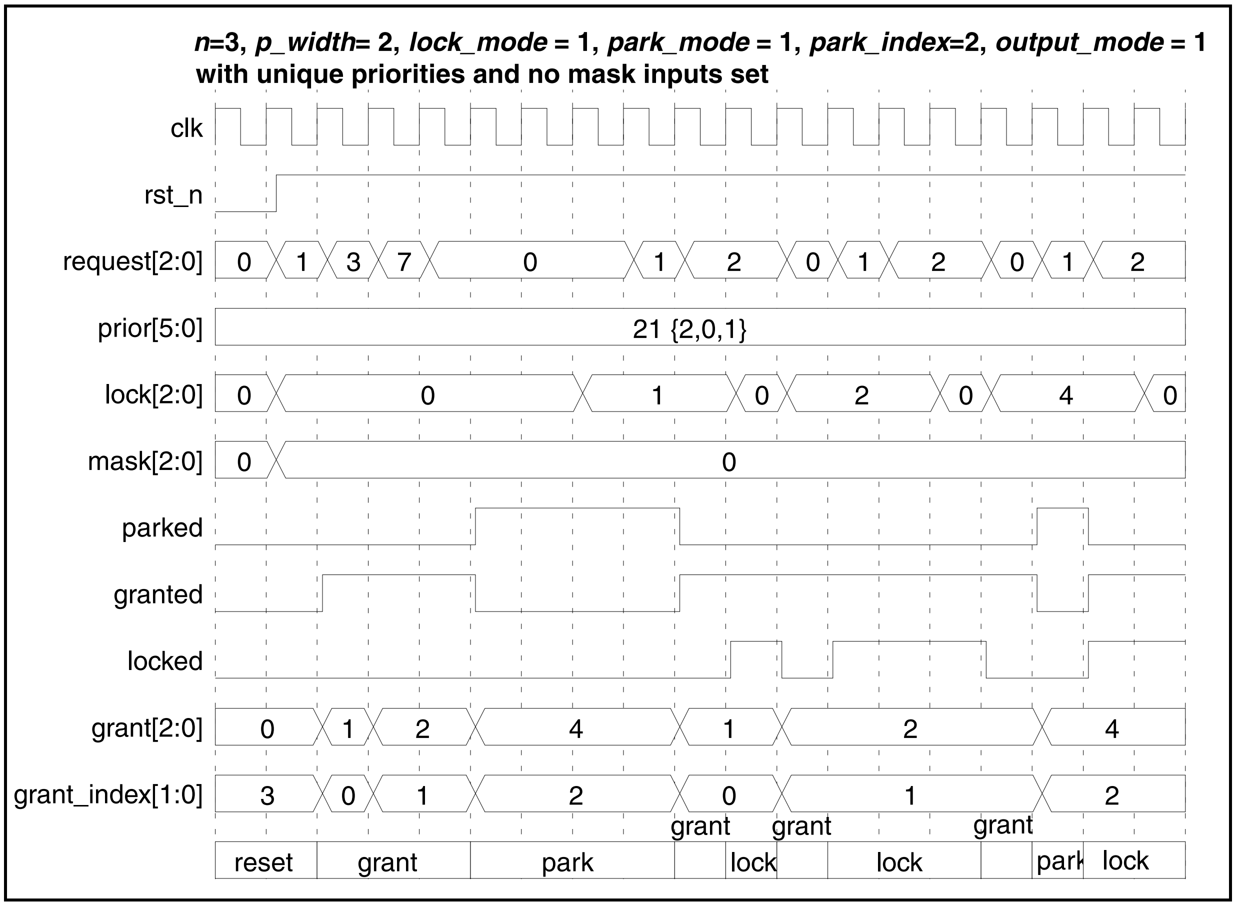
In the lock state the internal priorities of actively requesting inputs are held at levels they were prior to entering the lock state. But if any of the inputs, deassert the request in the lock state, its internal priority is set to the lowest level.

A potential deadlock condition could occur if two or more clients of the arbiter have the same programmed priority value and assert the request during the same cycle. In such cases, the DW_arb_2t uses the index of clients as a tie-breaker among the requesting clients. For example, the client connected to the input `request[0]` has the highest priority, while the client connected to `request[n-1]` has the lowest priority.

Timing Waveforms

The following figures shows timing diagrams for various conditions:

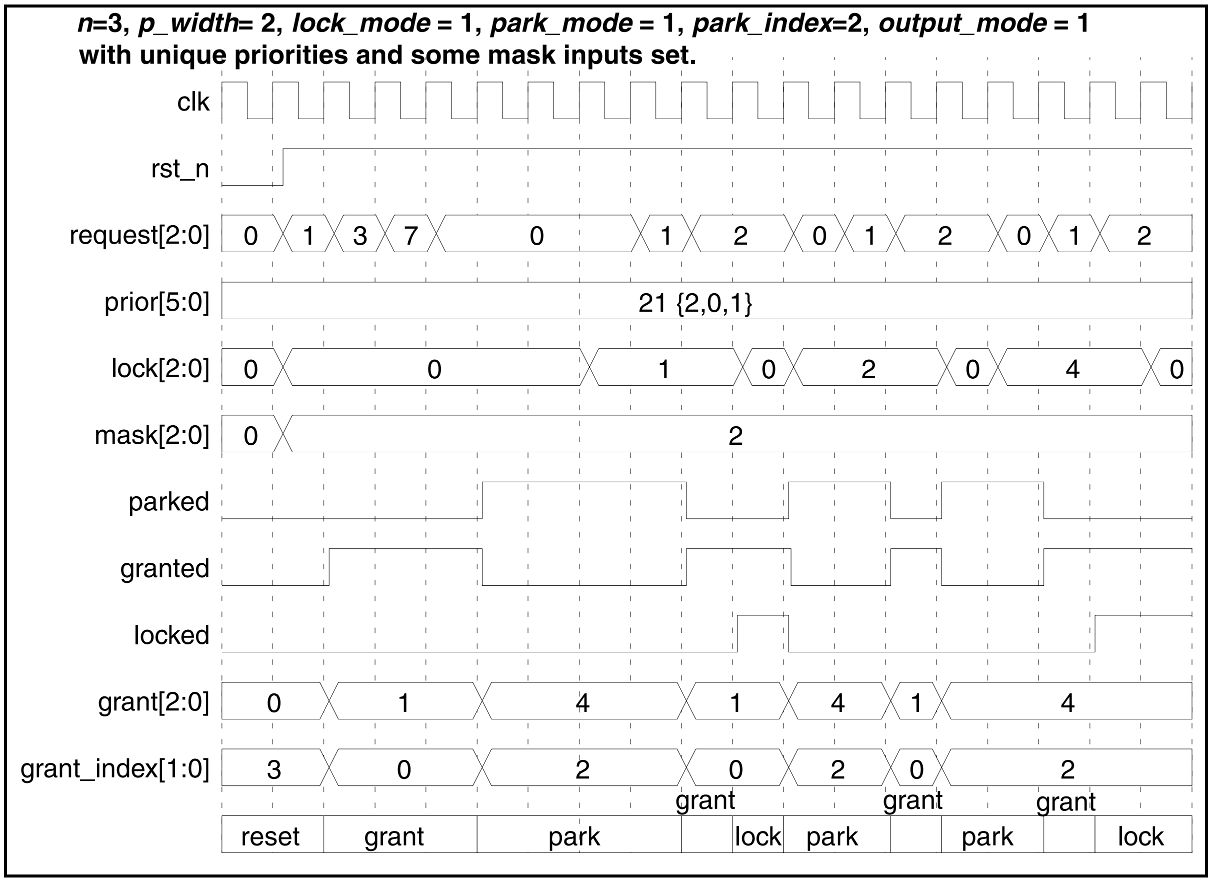
Figure 1-6 Waveform 1



The priorities of the three clients of the arbiter in the above mentioned case are `client[0]`=1, `client[1]`=0, and `client[2]`=2.

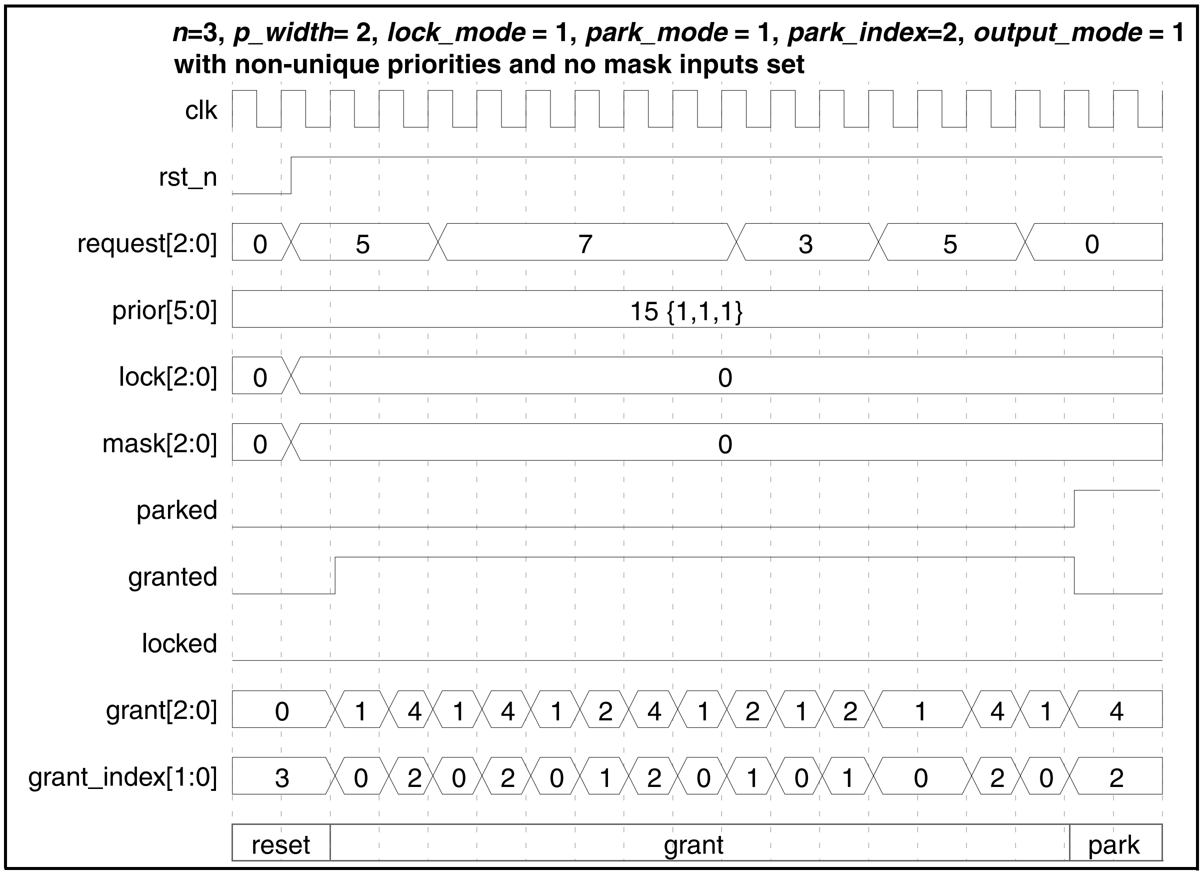
Any client can be masked off by setting the corresponding `mask` bit. By doing so it will not be considered for the arbitration. If `mask` bits are set and none of the non-masked clients are actively requesting, the arbiter will be parked to the designated client defined by *park_index*. In the non-locked state of the arbiter, setting the `mask` bit of the currently granted client effectively invalidates the request from the client. In the following cycle, the current `grant` is deasserted, and based on the current unmasked requests from other clients, a new client is generated. However, when a client has locked the arbiter, setting the `mask` bit of any client has no effect on the current grant.

Figure 1-7 Waveform 2



The priorities of the three clients of the arbiter in the above mentioned case are `client[0]=1`, `client[1]=0`, and `client[2]=2`.

Figure 1-8 Waveform 3



The priorities of the three clients of the arbiter in the above mentioned case are all equal - `client[0]=1`, `client[1]=1`, and `client[2]=1`.

Figure 1-9 Waveform 4

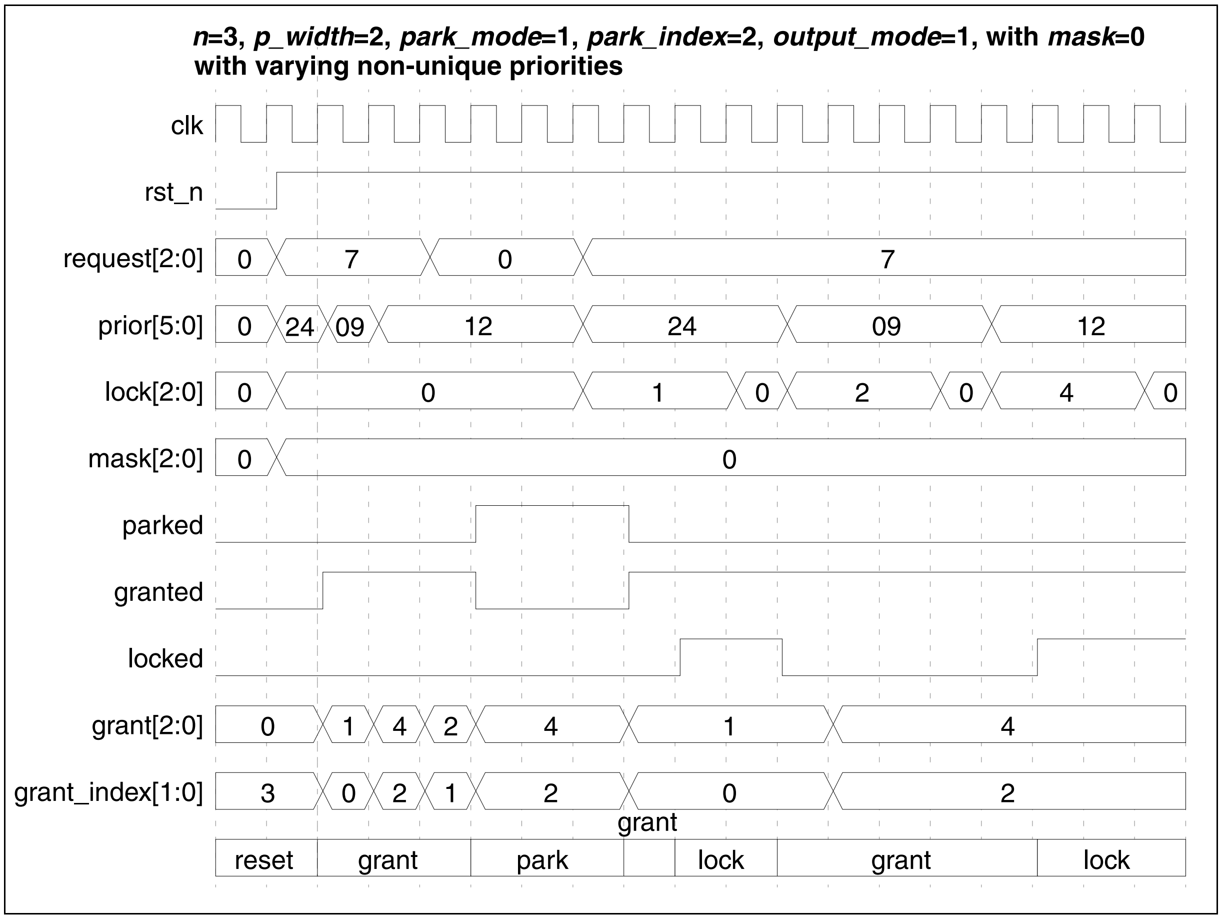
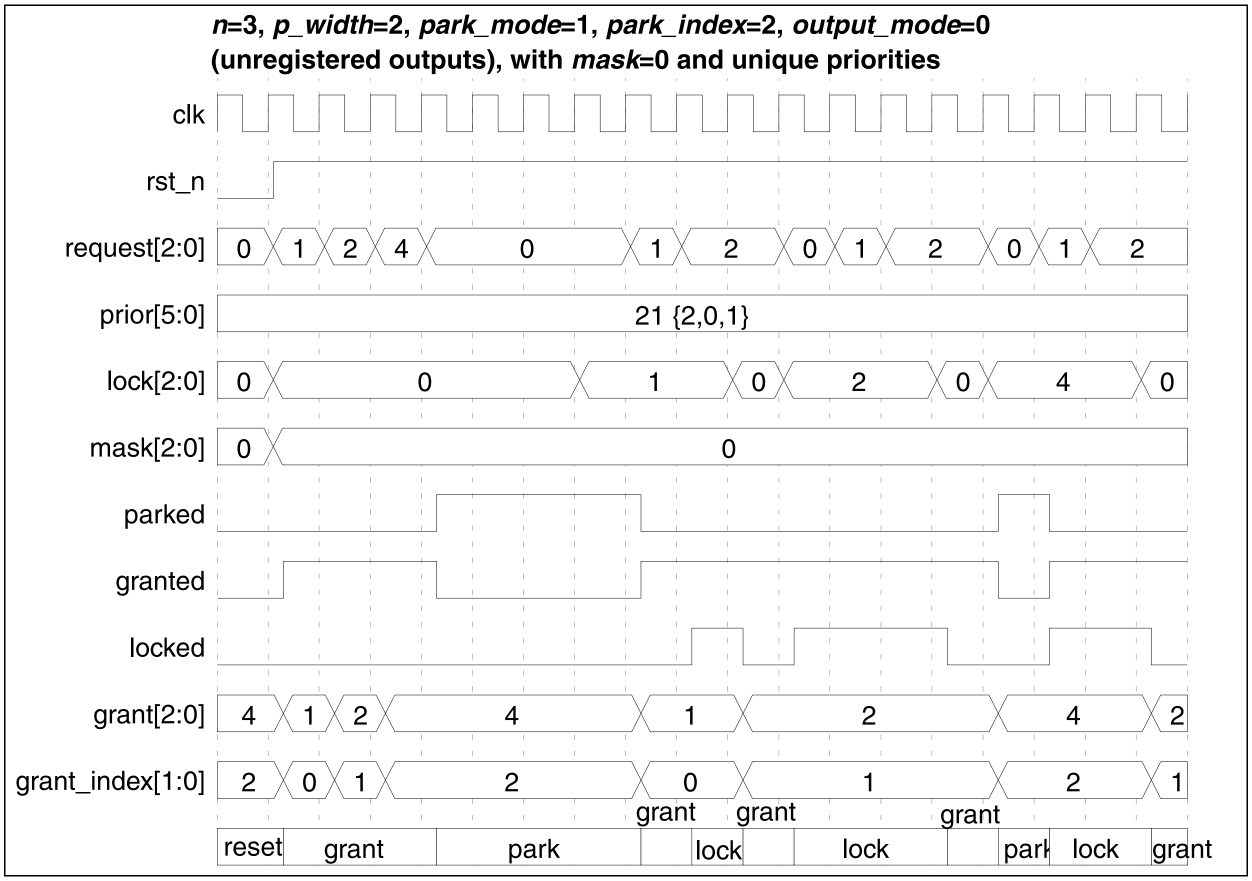


Figure 1-10 Waveform 5



Related Topics

- Application Specific – Control Logic Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.dw_foundation_comp.all;

entity DW_arb_2t_inst is
      generic (
        inst_n : NATURAL := 4;
        inst_p_width : NATURAL := 2;
        inst_park_mode : NATURAL := 1;
        inst_park_index : NATURAL := 0;
        inst_output_mode : NATURAL := 1
        );
      port (
        inst_clk : in std_logic;
        inst_rst_n : in std_logic;
        inst_init_n : in std_logic;
        inst_enable : in std_logic;
        inst_request : in std_logic_vector(inst_n-1 downto 0);
        inst_prior : in std_logic_vector(inst_p_width*inst_n-1 downto 0);
        inst_lock : in std_logic_vector(inst_n-1 downto 0);
        inst_mask : in std_logic_vector(inst_n-1 downto 0);
        parked_inst : out std_logic;
        granted_inst : out std_logic;
        locked_inst : out std_logic;
        grant_inst : out std_logic_vector(inst_n-1 downto 0);
       grant_index_inst : out std_logic_vector(bit_width(inst_n)-1
          downto 0)
        );
    end DW_arb_2t_inst;


architecture inst of DW_arb_2t_inst is

begin

    -- Instance of DW_arb_2t
    U1 : DW_arb_2t
    generic map (
          n => inst_n,
          p_width => inst_p_width,
          park_mode => inst_park_mode,
          park_index => inst_park_index,
          output_mode => inst_output_mode
          )
    port map (
          clk => inst_clk,
          rst_n => inst_rst_n,
          init_n => inst_init_n,
          enable => inst_enable,
          request => inst_request,
          prior => inst_prior,
          lock => inst_lock,
          mask => inst_mask,
          parked => parked_inst,
          granted => granted_inst,
          locked => locked_inst,
          grant => grant_inst,
          grant_index => grant_index_inst
          );


end inst;

-- pragma translate_off
configuration DW_arb_2t_inst_cfg_inst of DW_arb_2t_inst is
  for inst
  end for; -- inst
end DW_arb_2t_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_arb_2t_inst( inst_clk, inst_rst_n, inst_init_n, inst_enable,
          inst_request, inst_prior, inst_lock, inst_mask, parked_inst,
          granted_inst, locked_inst, grant_inst, grant_index_inst );

parameter inst_n = 4;
parameter inst_p_width = 2;
parameter inst_park_mode = 1;
parameter inst_park_index = 0;
parameter inst_output_mode = 1;

`define bit_width_n 2// bit_width_n is set to ceil(log2(n))

input inst_clk;
input inst_rst_n;
input inst_init_n;
input inst_enable;
input [inst_n-1 : 0] inst_request;
input [inst_p_width*inst_n-1 : 0] inst_prior;
input [inst_n-1 : 0] inst_lock;
input [inst_n-1 : 0] inst_mask;
output parked_inst;
output granted_inst;
output locked_inst;
output [inst_n-1 : 0] grant_inst;
output [`bit_width_n-1 : 0] grant_index_inst;

    // Instance of DW_arb_2t
    DW_arb_2t #(inst_n, inst_p_width, inst_park_mode, inst_park_index,
                      inst_output_mode) U1 (
                .clk(inst_clk),
                .rst_n(inst_rst_n),
                .init_n(inst_init_n),
                .enable(inst_enable),
                .request(inst_request),
                .prior(inst_prior),
                .lock(inst_lock),
                .mask(inst_mask),
                .parked(parked_inst),
                .granted(granted_inst),
                .locked(locked_inst),
                .grant(grant_inst),
                .grant_index(grant_index_inst) );

endmodule
```

Copyright Notice and Proprietary Information

© 2018 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com