The Abdus Salam International Centre for Theoretical Physics

2384-15

ICTP Latin-American Advanced Course on FPGADesign for Scientific Instrumentation

19 November - 7 December, 2012

Introduction to the WISHBONE Bus Interface

CICUTTIN Andres

ICTP Multidisciplinary Laboratory
Via Beirut 31
(34100) Trieste
ITALY

Introduction to

The WISHBONE Bus Interface

A System-on-chip (SoC)
Interconnection Architecture
For Portable IP Cores

Taken from the
WISHBONE SoC Architecture Specification, Revision B.3

What is ***Wishbone ?***

1) A general purpose interface between IP cores. It defines the standard data exchange between IP core modules.

2) A flexible design methodology for use with semiconductor IP cores.

3) “It is not a bus”

Some of the advantages

- **Promotes design reuse by alleviating system-on-chip integration problems**.
- **Improves the portability and reliability of the system**.
- Faster design time.
- Can be used for soft core, firm core or hard core IP.
- Does not require the use of specific development tools or target hardware.
- Fully compliant with virtually all logic synthesis tools.
- **Documentation standards simplify IP core reference manuals**.

WISHBONE Features I

- Simple, compact, logical IP core hardware interfaces requiring very few logic gates.
- **Supports structured design methodologies used by large project teams**.
- **Full set of popular data transfer bus protocols including:**
  - READ/WRITE cycle.
  - BLOCK transfer cycle.
  - RMW cycle.
- Modular data bus widths and operand sizes.
- Supports both *big endian* and *little endian* data ordering.

WISHBONE Features II

- **Supports variable core interconnection methods:**
  Point-to-point, shared bus, crossbar switch, and switched fabric interconnections.
- **Handshaking protocol allows each IP core to throttle its data transfer speed**.
- **Supports single clock data transfers**.
- Supports normal cycle termination, retry termination and termination due to error.
- Modular address widths.
- **Partial address decoding scheme for slaves**.

WISHBONE Features III

- User-defined tags. These are useful for applying information to an address bus, a data bus or a bus cycle.
- MASTER / SLAVE architecture for very flexible system designs.

WISHBONE Features IV

- Multiprocessing (*multi-MASTER*) capabilities. This allows for a wide variety of System-on-Chip configurations.
- **Arbitration methodology is defined by the end user** (*priority arbiter, round-robin arbiter,* etc.).
- **Supports various IP core interconnection means, including:**
  - Point-to-point
  - Shared bus
  - Crossbar switch
  - Data flow interconnection
- **Synchronous design** (Assures portability, and ease of use)

Specification Keywords

1. **RULES**
2. **RECOMMENDATIONS**
3. **SUGGESTIONS**
4. **PERMISSIONS**
5. **OBSERVATIONS**

RULES

Rules form the basic framework of the specification. Rules are characterized by an imperative style. The uppercase words MUST and MUST NOT are reserved exclusively for stating rules.

RECOMMENDATIONS

Whenever a recommendation appears, designers would be wise to take the advice given. Doing otherwise might result in some awkward problems or poor performance. These are provided as guidance for the user.

SUGGESTIONS

A suggestion contains advice which is helpful but not vital. The reader is encouraged to consider the advice before discarding it. Some design decisions are difficult until experience has been gained. Some suggestions have to do with designing compatible interconnections, or with making system integration easier.

PERMISSIONS

In some cases a rule does not specifically prohibit a certain design approach, but the reader might be left wondering whether that approach might violate the spirit of the rule, or whether it might lead to some subtle problem. Permissions reassure the reader that a certain approach is acceptable and will not cause problems. The upper-case word MAY is reserved exclusively for stating a permission and is not used for any other purpose.

OBSERVATIONS

Observations do not offer any specific advice. They usually clarify what has just been discussed. They spell out the implications of certain rules and bring attention to things that might otherwise be overlooked. They also give the rationale behind certain rules, so that the reader understands why the rule must be followed.

Examples: Rules and Permissions

- **RULE 3.00**
  All WISHBONE interfaces **MUST** initialize themselves at the rising [CLK_I] edge following the assertion of [RST_I]. They **MUST** stay in the initialized state until the rising [CLK_I] edge that follows the negation of [RST_I].

- **RULE 3.05**
  [RST_I] **MUST** be asserted for at least one complete clock cycle on all WISHBONE interfaces.

- **PERMISSION 3.00**
  [RST_I] **MAY** be asserted for more than one clock cycle, and **MAY** be asserted indefinitely.

- **RULE 3.10**
  All WISHBONE interfaces **MUST** be capable of reacting to [RST_I] at any time.

- **RULE 3.15**
  All self-starting state machines and counters in WISHBONE interfaces **MUST** initialize themselves at the rising [CLK_I] edge following the assertion of [RST_I]. They **MUST** stay in the initialized state until the rising [CLK_I] edge that follows the negation of [RST_I].

Examples: Recomendations, suggestions and observations

- **RECOMENDATION 3.00**

  Design SYSCON modules so that they assert [RST_O] during a power-up condition. [RST_O] **should** remain asserted until all voltage levels and clock frequencies in the system are stabilized. When negating [RST_O], **do so** in a synchronous manner that conforms to this specification.

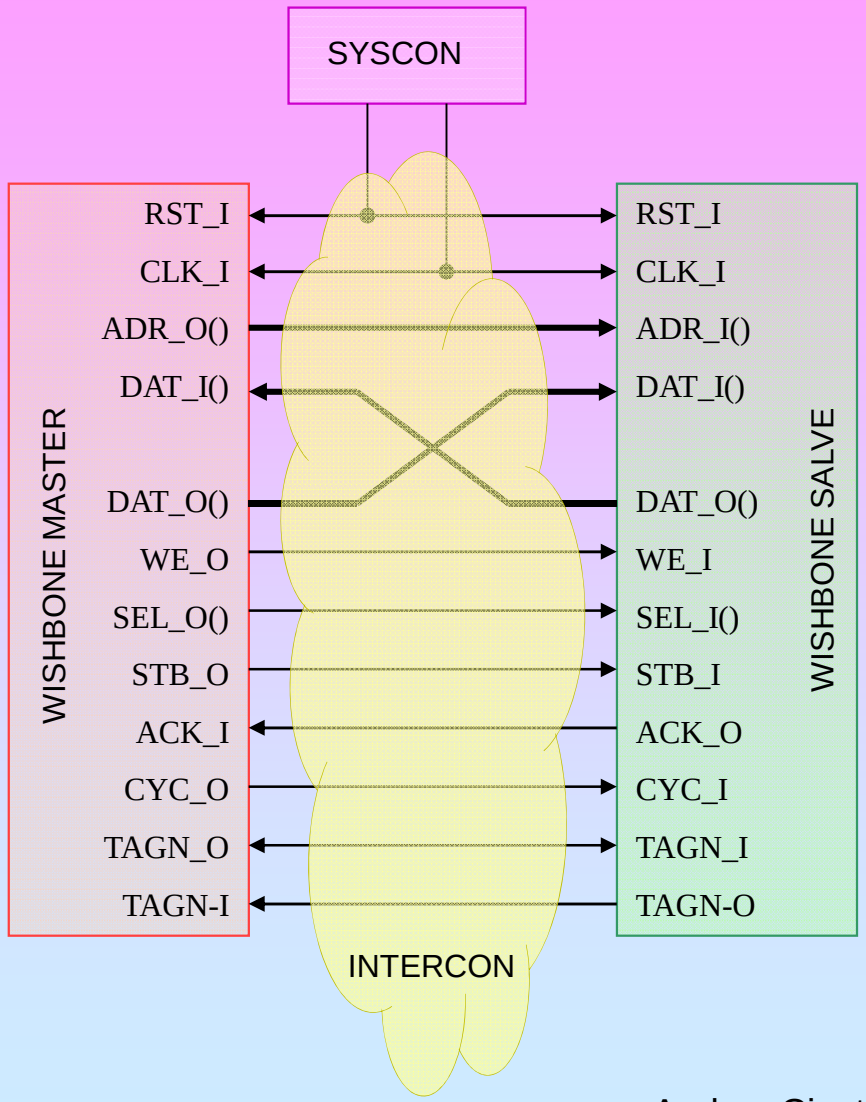
- **SUGGESTION 3.00**

  Some circuits require an *asynchronous* reset capability. If an IP core or other SoC component requires an asynchronous reset, then **define** it as a non-WISHBONE signal. This prevents confusion with the WISHBONE reset [RST_I] signal that uses a purely synchronous protocol, and needs to be applied to the WISHBONE interface only.

- **OBSERVATION 3.20**

  All WISHBONE *interfaces* respond to the reset signal. However, the IP Core connected to a WISHBONE interface does not necessarily need to respond to the reset signal.

The Wishbone Modules

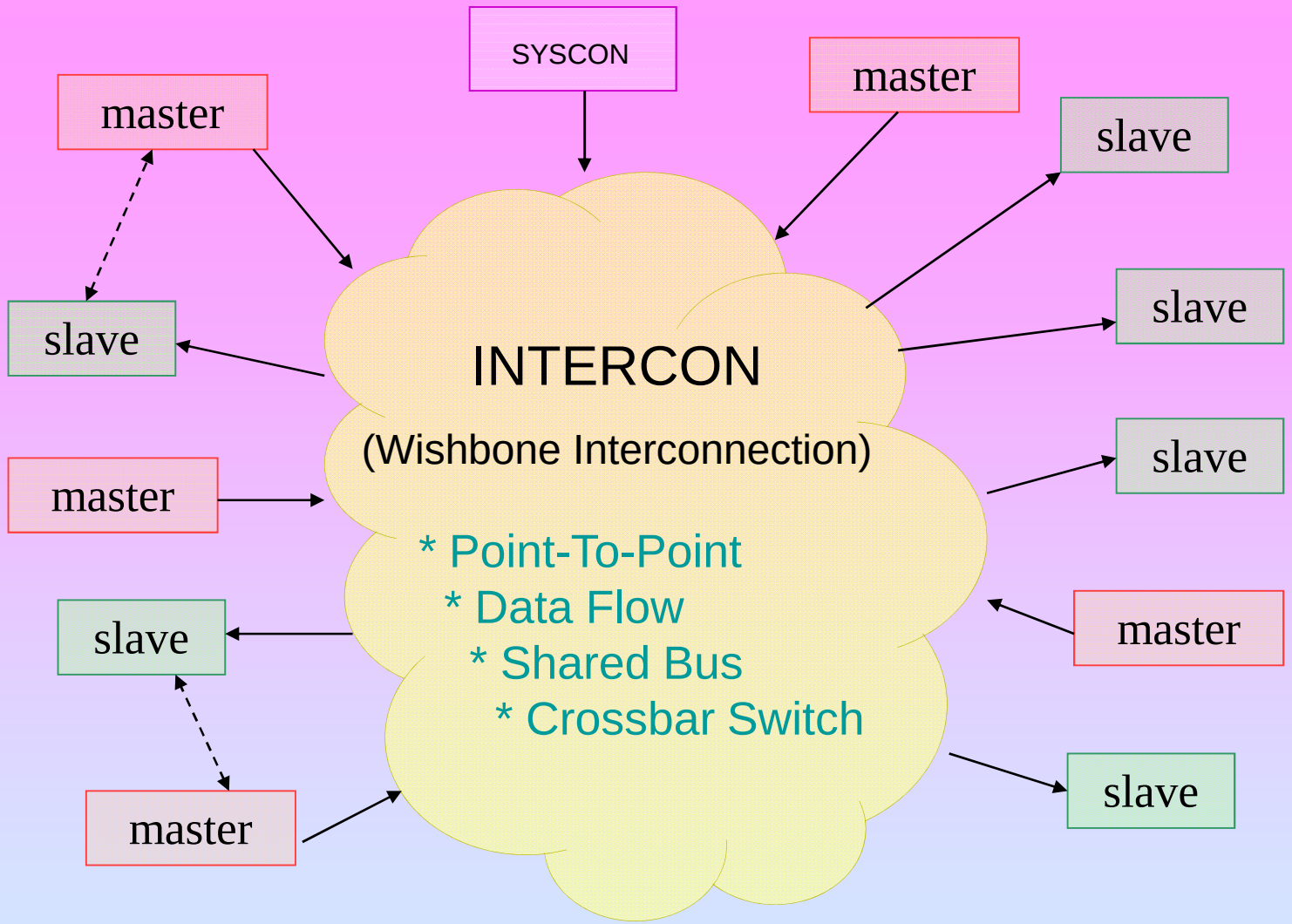


SYSCON: drives the system clock [CLK_O] and reset [RST_O] signals.

MASTER: IP Core interface that generates bus cycles.

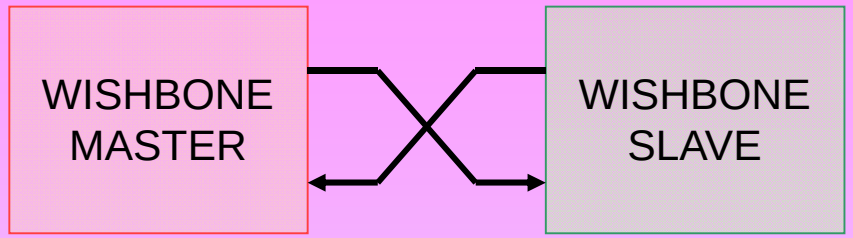
SLAVE: IP Core interface that receives bus cycles.

INTERCON: an IP Core that connects all of the MASTER and SLAVE interfaces together.

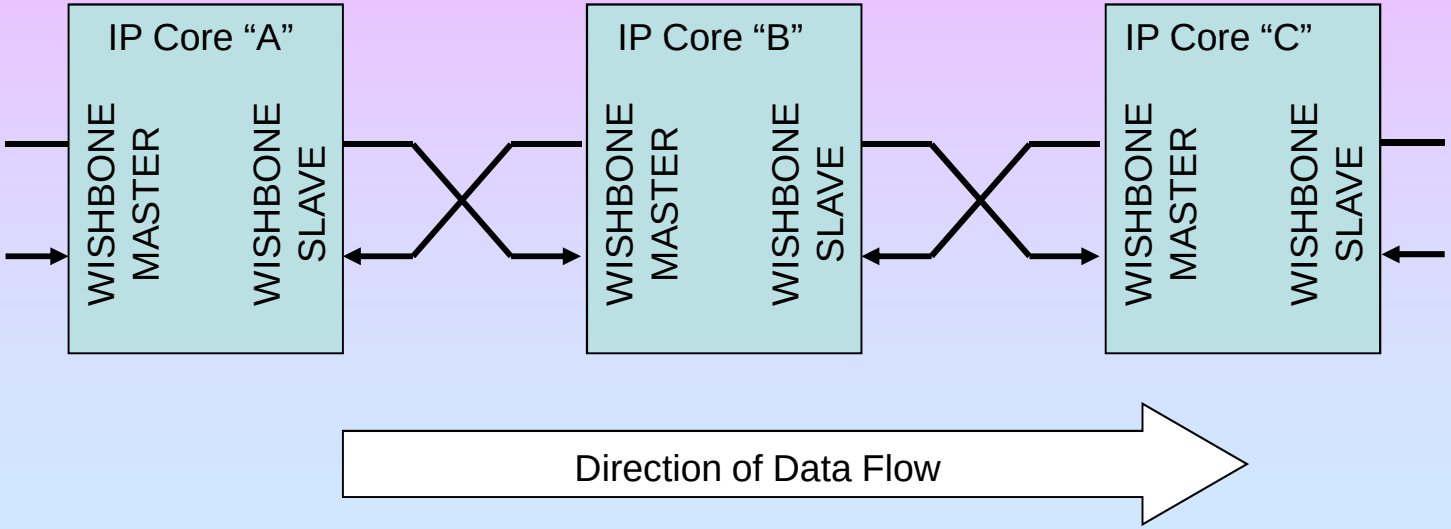


The Wishbone Interconnection is created by the SYSTEM INTEGRATOR, who has total control of its design!

Interconnections

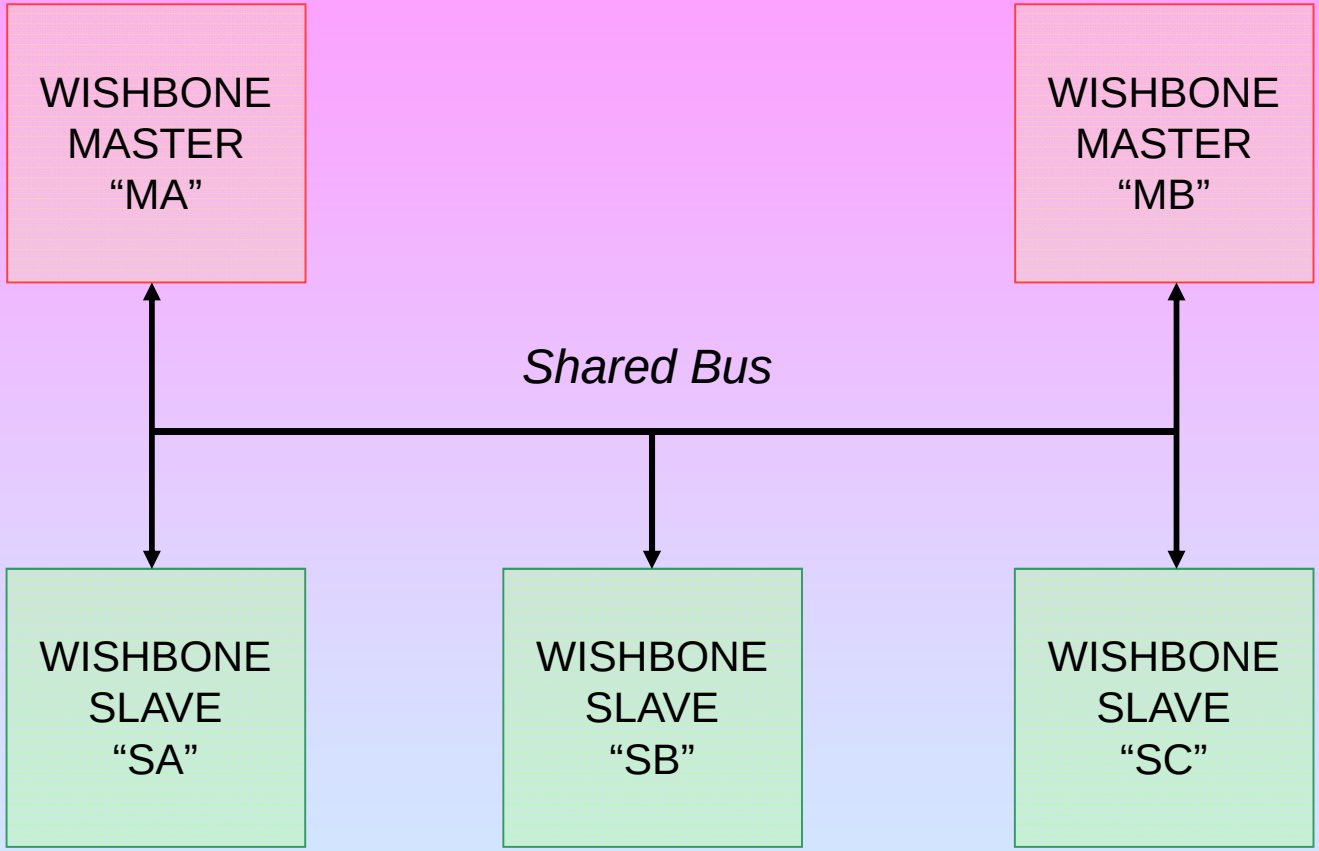


Point-To-Point Interconnection



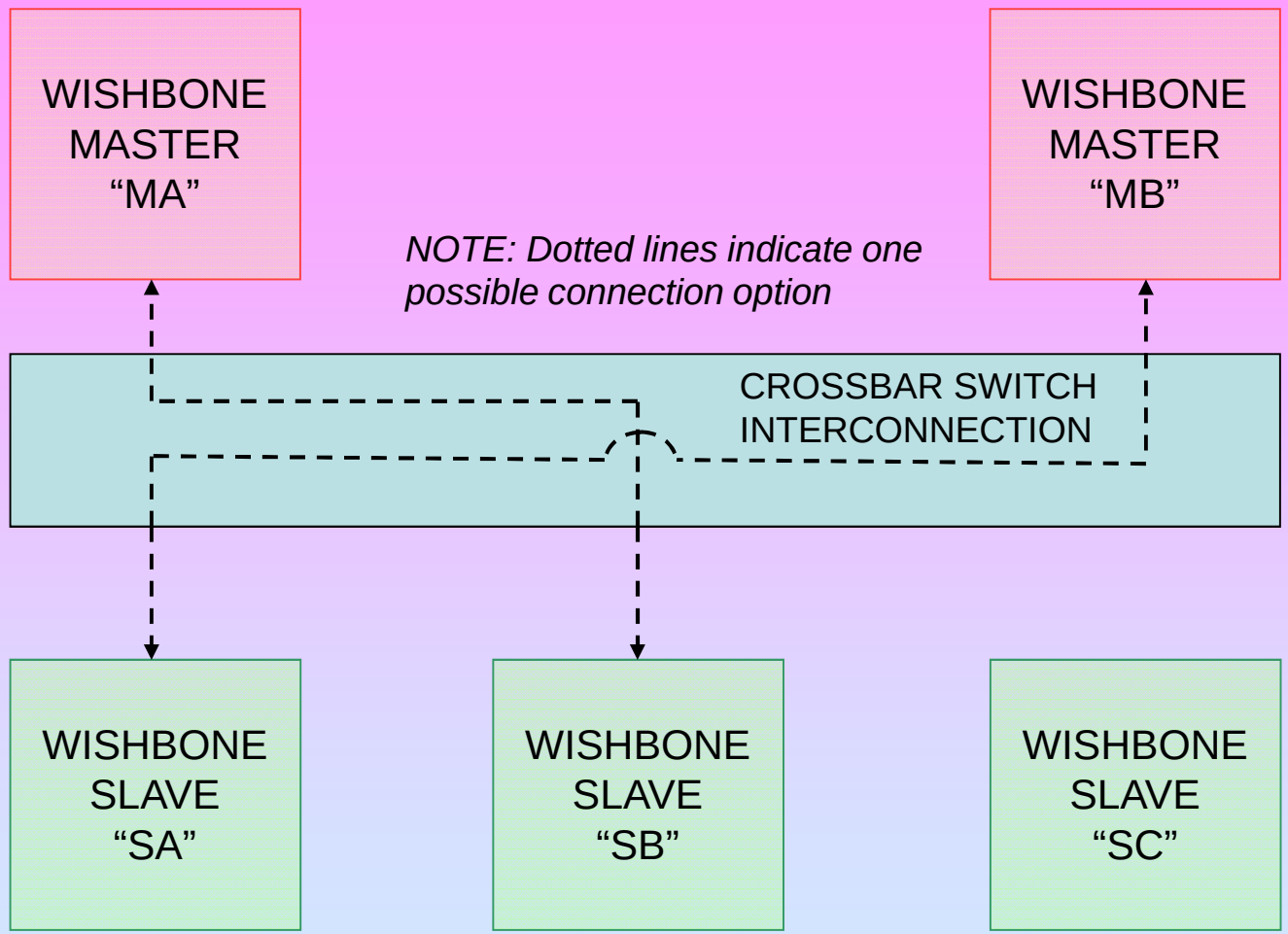
Data Flow Interconnection

Interconnections



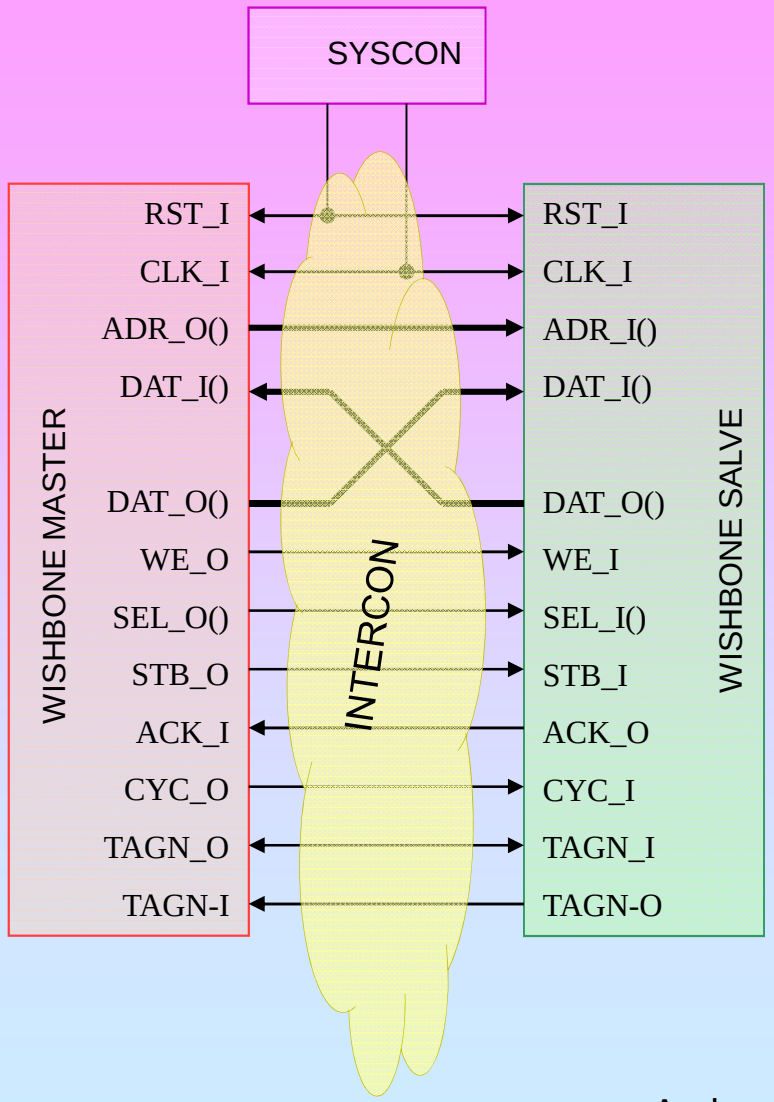
Shared Bus Interconnection

Interconnections



Crossbar Switch Interconnection

The Main Wishbone Signals *(Ports)*



RST_I: receives the reset output signal [RST_O] from the SYSCON.

CLK_I: receives the clock output signal [CLK_O] from the SYSCON.

ADR_I: receives the address from MASTER

ADR_O: drives the address to the SLAVE

DAT_I: receives the data

DAT_O: drives the data

WE_I, WE_O: Write Enable (*active high*)

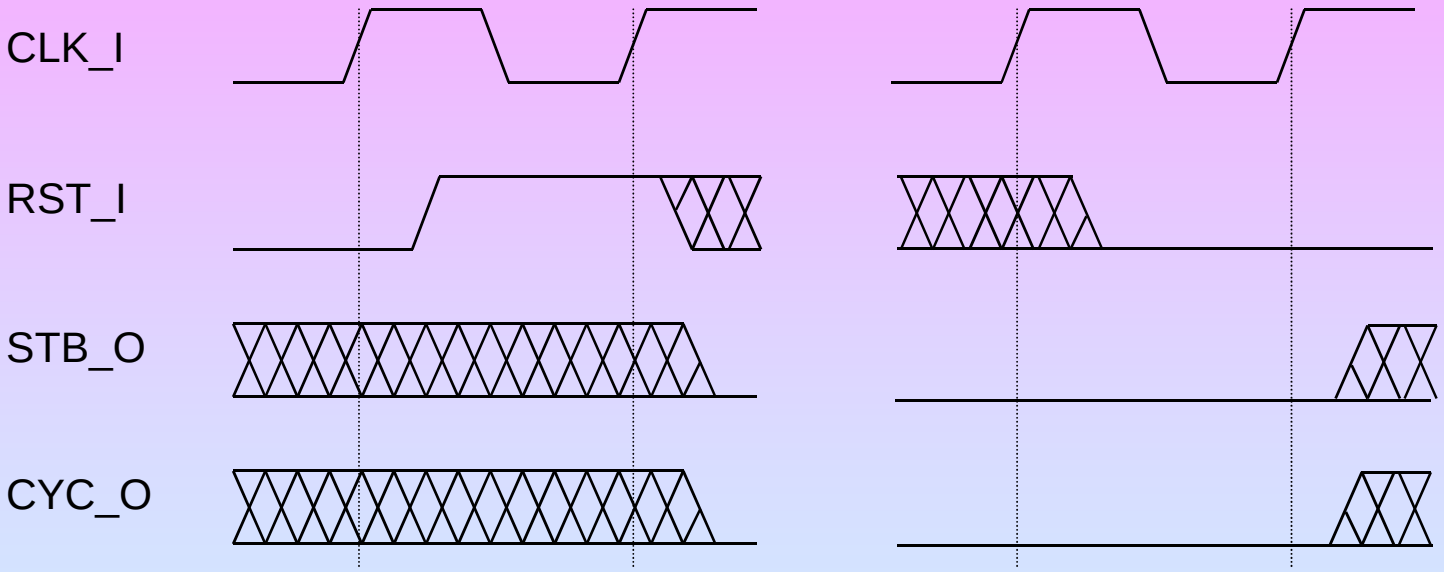
STB_I, STB_O: Strobe (*kind of chip select*)

ACK_I, ACK_O: Acknowledge (*active high*)

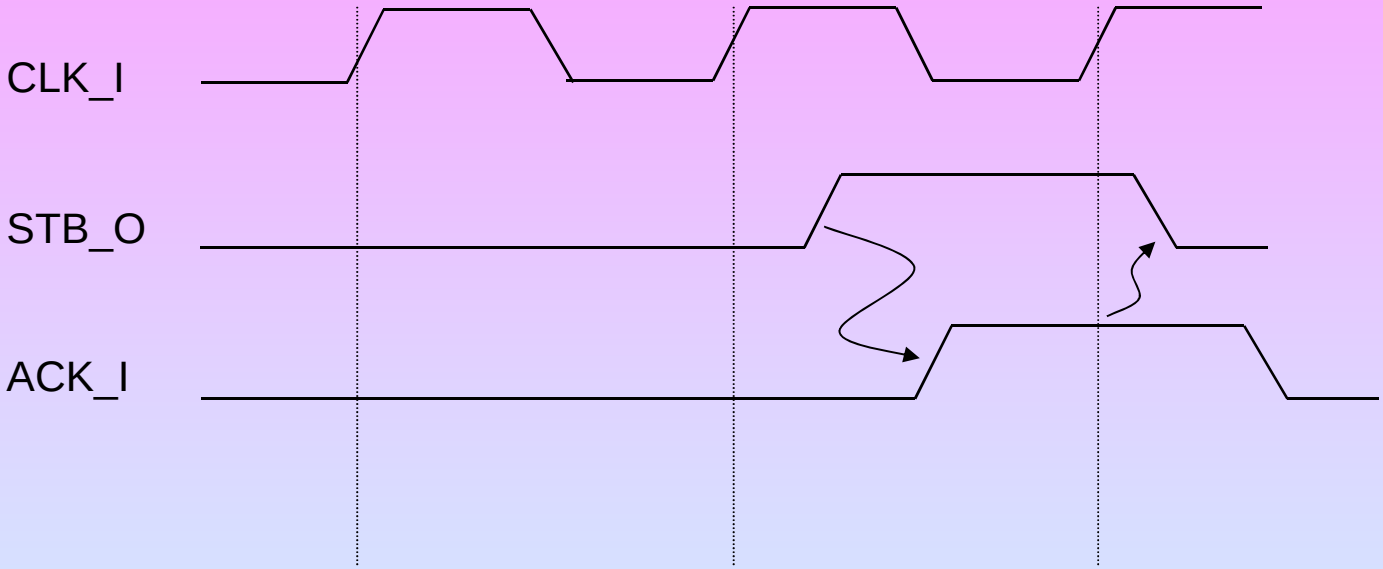
CYC_I, CYC_O: Bus Cycle (*active high*)

WISHBONE Classic Bus Cycles

The Reset Cycle



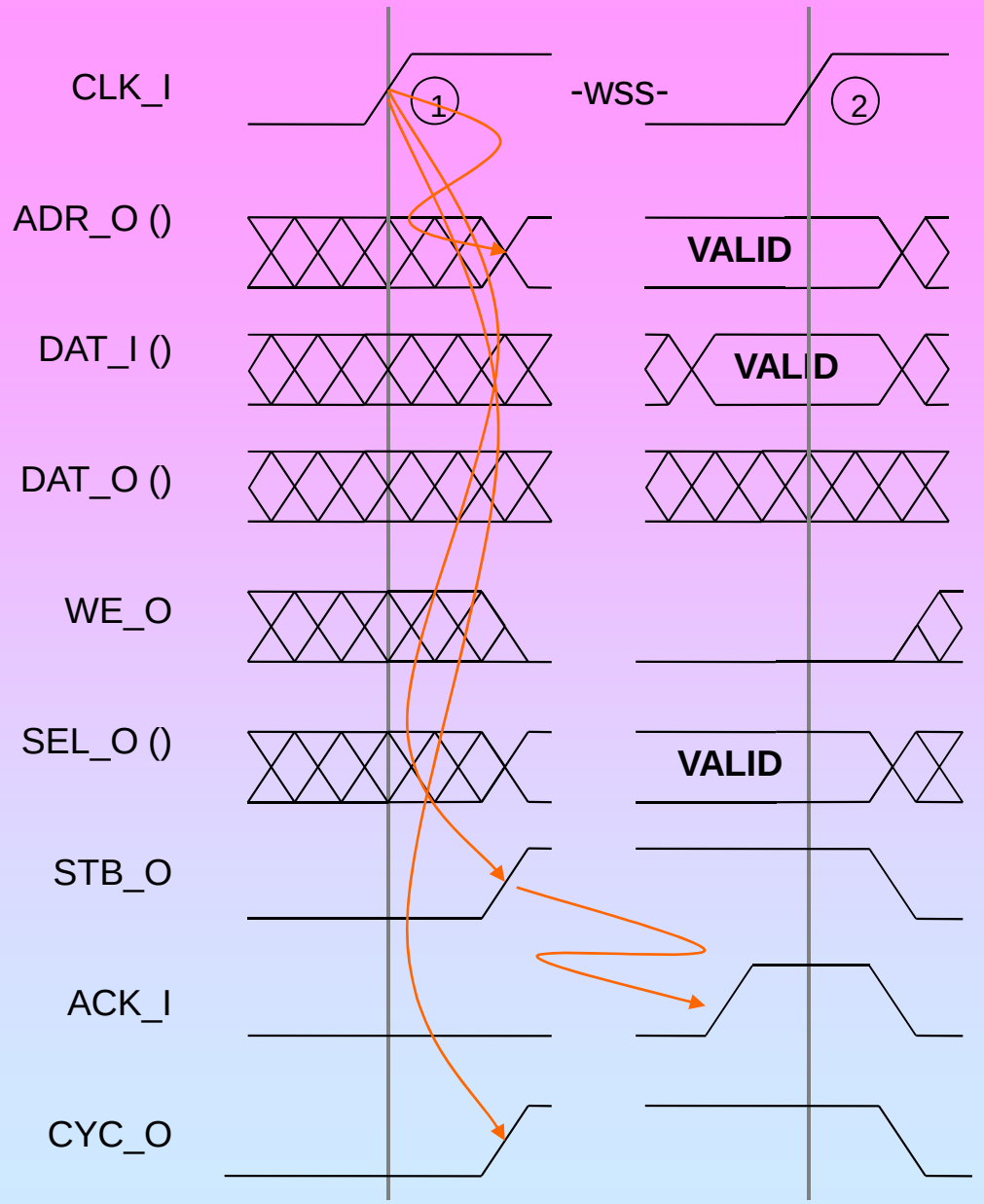
Local Bus Handshaking Protocol



Tag Types

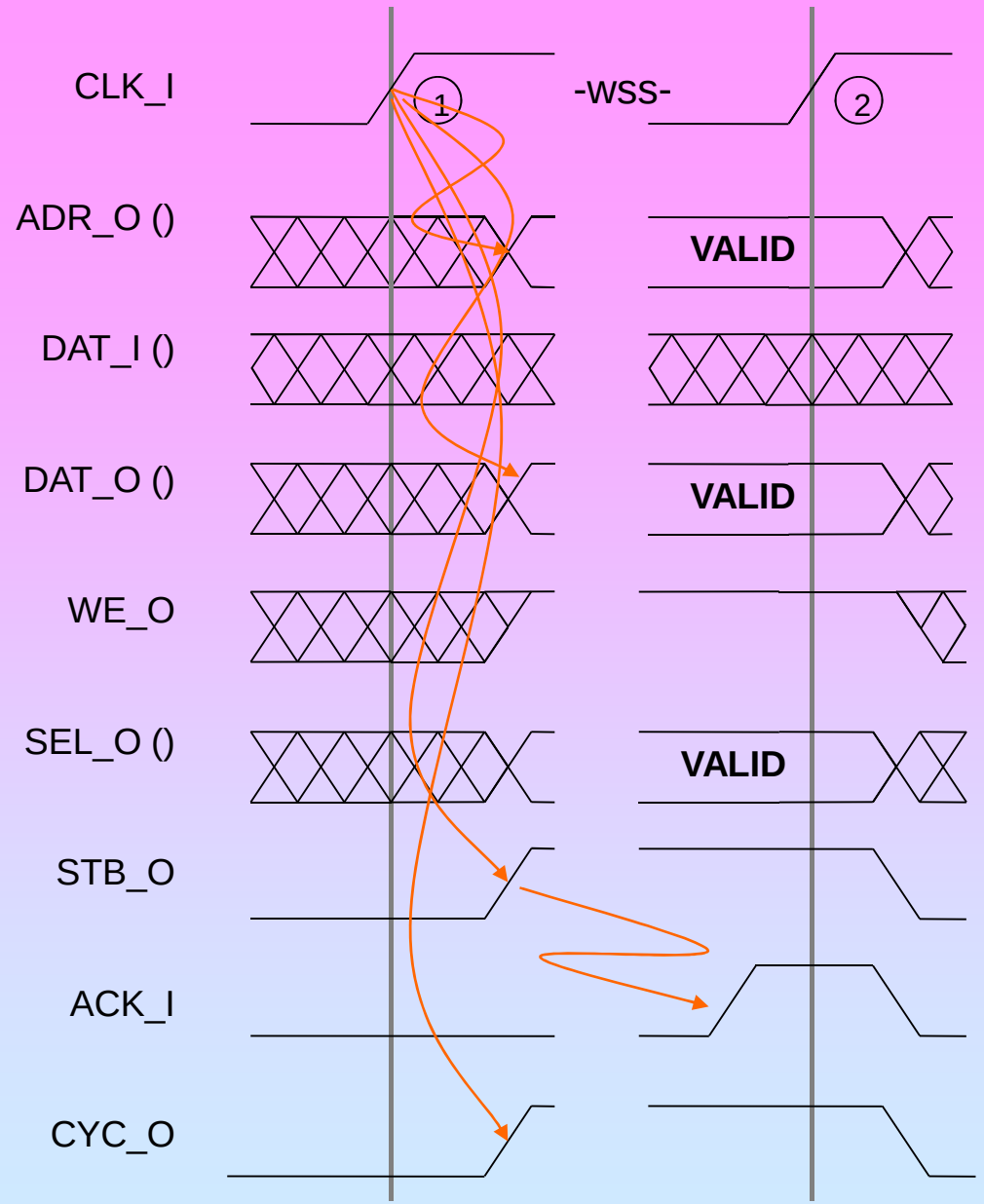
	Master		Slave	
Description	TAG TYPE	Associated with	TAG TYPE	Associated with
Address tag	TGA_O()	ADR_O	TGA_I()	ADR_I()
Data tag input	TGD_I()	DAT_I()	TGD_I()	DAT_I()
Data tag output	TGD_O()	DAT_O()	TGD_O()	DAT_O()
Cycle tag	TGC_O()	Bus Cycle	TGC_I()	Bus Cycle

SINGLE READ cycle
(Master Signals)

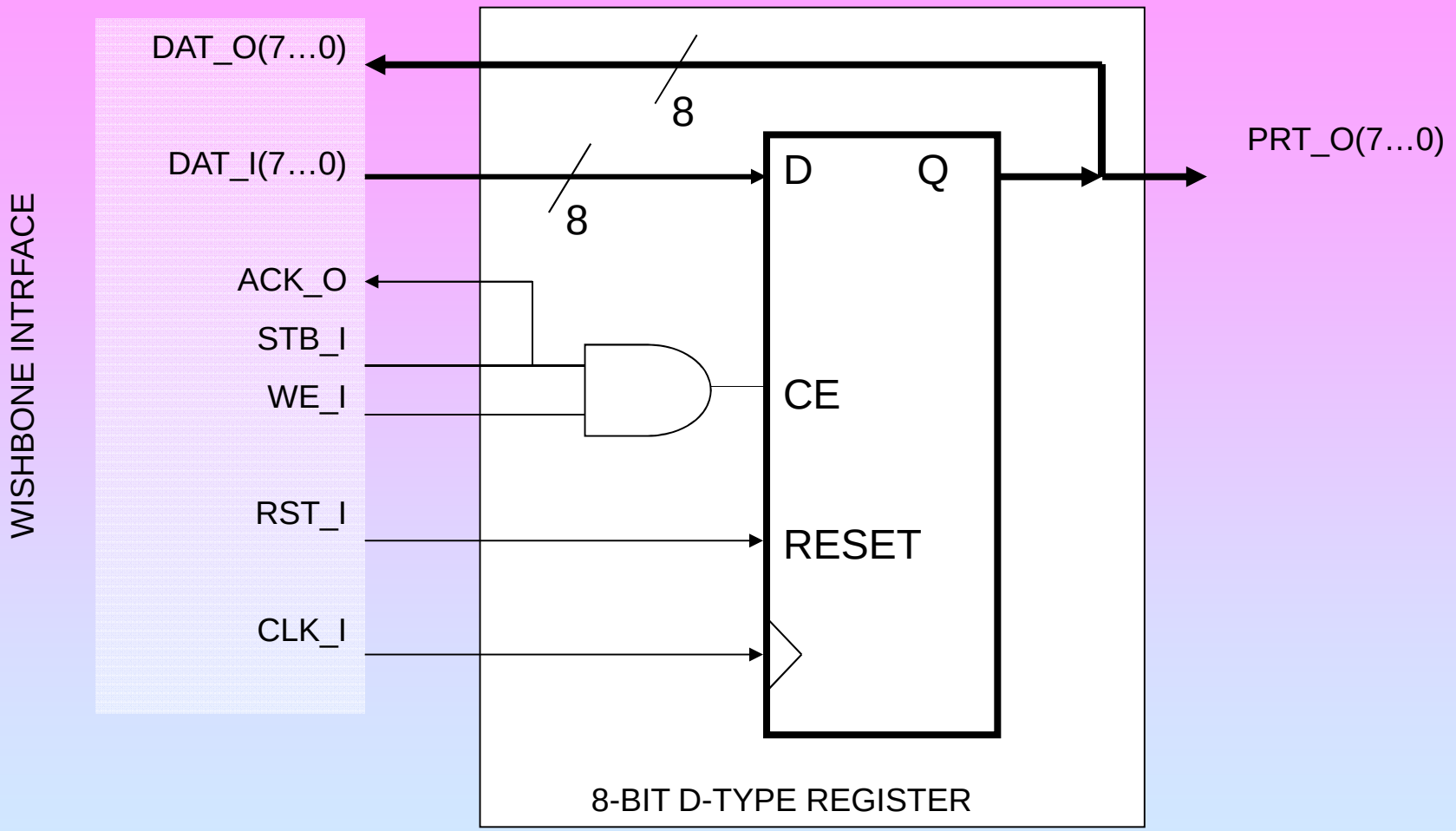


SINGLE WRITE cycle

(Master Signals)



A Simple Example: WISHBONE SLAVE port



VHDL implementation of the 8-bit output port interface (*pag. 109*)

```
library ieee;
use ieee.std_logic_1164.all;

entity WBOPRT08 is
port(
      -- WISHBONE SLAVE interface:

                ACK_O: out std_logic;
                CLK_I: in std_logic;
                DAT_I: in std_logic_vector( 7 downto 0 );
                DAT_O: out std_logic_vector( 7 downto 0 );
                RST_I: in std_logic;
                STB_I: in std_logic;
                WE_I: in std_logic;

     -- Output port (non-WISHBONE signals):

                PRT_O: out std_logic_vector( 7 downto 0)
   );

end entity WBOPRT08;
```

```
architecture WBOPRT081 of WBOPRT08 is

        signal Q: std_logic_vector( 7 downto 0 );

Begin

internal_DAT_I <= DAT_I;

REG: process( CLK_I )
    begin
       if( rising_edge( CLK_I ) ) then
          if( RST_I = '1' ) then     --synchronous reset
                Q <= B"00000000";
          elsif( (STB_I and WE_I) = '1' ) then
                Q <= internal_DAT_I( 7 downto 0 );
          else
                Q <= Q;
          end if;
       end if;
end process REG;

ACK_O <= STB_I;  --(Asynchronous assignments !)
DAT_O <= Q;
PRT_O <= Q;

end architecture WBOPRT081;
```

WB Clasic Bus Cycles

Other WB Bus Cycles (WISHBONE Registered Feedback)

Both Master and Slave MUST support "Cycle Type Identifier": [CTI_O()], [CTI_I()]

4.2 Signal Description

CTI_IO()
The Cycle Type Identifier [CTI_IO()] Address Tag provides additional information about the current cycle. The MASTER sends this information to the SLAVE. The SLAVE can use this information to prepare the response for the next cycle.

PERMISSION 4.05
MASTER and SLAVE interfaces MAY be designed to support the [CTI_I()] and [CTI_O()] signals. Also MASTER and SLAVE interfaces MAY be designed to support a limited number of burst types.

RULE 4.05
MASTER and SLAVE interfaces that do support the [CTI_I()] and [CTI_O()] signals MUST at least support the Classic cycle [CTI_IO()='000'] and the End-of-Cycle [CTI_IO()='111'].

RULE 4.10
MASTER and SLAVE interfaces that are designed to support a limited number of burst types MUST complete the

Table 4-2 Cycle Type Identifiers

CTI_O(2:0)	Description
'000'	Classic cycle.
'001'	Constant address burst cycle
'010'	Incrementing burst cycle
'011'	*Reserved*
'100'	*Reserved*
'101	*Reserved*
'110'	*Reserved*
'111'	End-of-Burst

Other WB Bus Cycles

Classic Cycle *(with wait states –ws-)* *(Page 77)*

Constant Address Burst *(Page 83)*

(e.g. for fifos, ports, etc)

Incrementing Burst Cycle *(Page 86)*

An incrementing burst is defined as multiple accesses to consecutive addresses. Each transfer the address is incremented. The increment is dependent on the data array [DAT_O()], [DAT_I()] size; for an 8bit data array the increment is 1, for a 16bit data array the increment is 2, for a 32bit data array the increment is 4, etc.

All the information related to the

Wishbone SoC Architecture Specification

is contained in its official documentation:

wbspc_b3.pdf

- It is very clear and easy to read and understand
- It includes design examples
- The specification doesn't infringe patents, copyright, trademarks or trade secret rights of others

Conclusion

1. There are many and very good reasons to adopt the **Wishbone SoC Architecture Specification** as the standard interface for open-source projects such as the **Reconfigurable Virtual Instrumentation** project proposed by ITCP.
2. Your are strongly encouraged to use it to develop reusable IP blocks that can be easily shared by a large community of users and developers