

# Chapter 2

## Why use the IEEE 1500 Standard?



### 2.1 Introduction

Before the question of “Why use the IEEE 1500 Standard?” is asked, another question should be addressed. Why do we need a wrapper at all? A wrapper is an isolation boundary between a core and the rest of the design. This isolation boundary allows test reuse to occur on the core as well as the capability to fully test the logic external to the core without having to exercise or have access to the core. Other advantages, for which the wrapper allows, are modular or partition-based debug, diagnosis and testing.

#### 2.1.1 Test Reuse and Partitioning

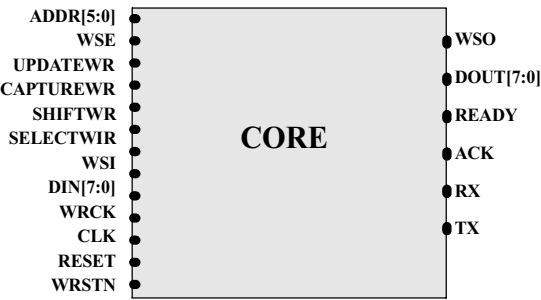
When an SOC is created, it can be a massive effort to generate the test patterns for screening the device. This effort cannot begin until very late in the design flow as the SOC must be complete first. In an SOC that is composed of wrapped cores, this effort can be divided between different people and done at different times.

Each core is probably completed at a different time. If the delivered core is wrapped, an isolated set of test patterns can be generated for it at the time it is completed. These patterns are delivered with the core and can be post-processed to run at the SOC level. This partitioning of tasks means less work in the critical path at the end of the SOC design.

A hard core can also be delivered by another company. A wrapper allows the core to be tested with a minimum pinset. In addition, the wrapper can be used to test logic external to the core. This is especially important if the core user does not have access to the netlist of the hard core. A model of the wrapper can be created that does not infringe on any of the core IP, but it is enough information to read into a tool to create control and observe logic external to the core.

This parallel design strategy can shorten the design schedule. If a design change is needed in a core, it can be done locally. The timing and layout does not need to be redone for the entire design and test generation needs to be redone for the changed core only.

Consider the wrapped core and corresponding pattern in Figure 2. Note the port names on the core and in the pattern set. Once this core is embedded into an SOC, the core signals may change names, disappear or new signals may need to be added. The core and core pattern can be reused in many designs relieving some of the workload required for an SOC design.

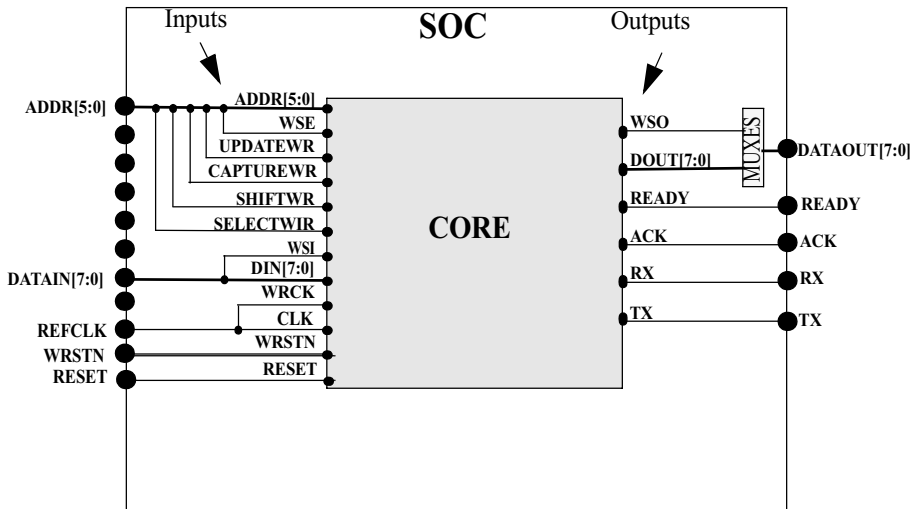


Core Pattern

W	W	U	C	S	S	W	C	R	W
R	S	P	A	H	E	S	R	L	E
C	I	D	P	I	L	E	S	K	S
K	A	T	F	E	T	E			
		T	U	T	C	N	T		
		E	R	W	T				
		W	E	R	W				
		R	W	I					
		R		R					
1	0	0	0	0	0	0	0	1	1
1	0	0	0	0	1	0	0	1	1
1	1	0	0	1	1	0	1	1	0
1	1	0	0	1	1	0	1	1	0
1	0	0	0	1	1	0	1	1	0
1	0	0	0	1	1	0	1	1	0
1	1	0	0	1	1	0	1	1	0
1	1	0	0	1	1	0	1	1	0
1	0	0	0	1	1	0	1	1	0

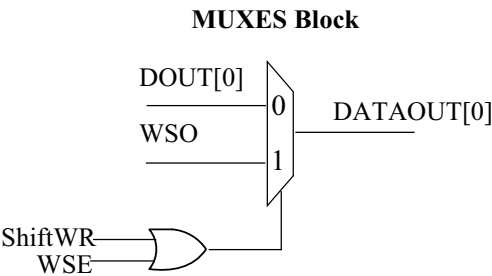
Figure 2 Isolated Core with Pattern

As shown in Figure 2, the test pattern is limited to the test signals only. Constraining the functional signals to X or removing them during test pattern generation is common practice.



**Figure 3** Core Embedded in an SOC

Figure 3 shows how the example wrapped core might be embedded into an SOC. All inputs are shown on the left side of the core and the outputs are shown on the right side of the core. Note that the SOC inputs, such as ADDR, can drive both the test inputs and the functional inputs on the core. These functional ports are not utilized during test as they are controlled and observed by the WBR. It is good practice to gate the test inputs off during functional mode - this is not shown in Figure 3. Also note that there is a multiplexer block (MUXES) connected to the WSO and DOUT outputs in Figure 3. The MUXES block determines which output ports of the core will be accessible from the DATAOUT pins. Figure 4 (page 16) shows what is inside the MUXES block. While the WSE or ShiftWR signal is high, the WSO signal is output to the DATAOUT[0] pin. While the WSE and ShiftWR signals are low, the DOUT[0] signal is output to the DATAOUT[0] pin. WSE and ShiftWR need to be held low (disabled) during functional mode – this is not shown in Figure 3.



**Figure 4** MUXES Block

Table 1 lists each core port and the SOC pin to which it is connected.

**Table 1** Core Port to SOC Connection

Core Port	SOC Pin	Direction	Type
WRCK	REFCLK	Input	Test
WSI	DATAIN[0]	Input	Test
UPDATEWR	ADDR[0]	Input	Test
CAPTUREWR	ADDR[1]	Input	Test
SHIFTWR	ADDR[2]	Input	Test
SELECTWIR	ADDR[3]	Input	Test
WRSTN	WRSTN	Input	Test
WSE	ADDR[4]	Input	Test
DIN[7:0]	DATAIN[7:0]	Input	Func
ADDR[5:0]	ADDR[5:0]	Input	Func
READY	READY	Output	Func
CLK	REFCLK	Input	Func
RESET	RESET	Input	Func
WSO	DATAOUT[0]	Output	Test
DOUT[7:0]	DATAOUT[7:0]	Output	Func
ACK	ACK	Output	Func
RX	RX	Output	Func
TX	TX	Output	Func

If patterns are created for a hard core that does not have a wrapper, each signal in the pattern must be ported out to an external pin in order to control all of the inputs and observe all of the outputs. However, if the core is wrapped, only the dedicated test and minimal functional pins (e.g. clocks and asynchronous resets) need to be ported to the top level of the SOC.

After the core is embedded, the test pattern delivered with the core must be manipulated to be run from the SOC level. Figure 5 shows how the core pattern from Figure 2 (page 14) is converted so that it is able to run from the SOC level. The names of each signal has been changed to accommodate the connections of the core pins to the SOC level.

In other words though the data is the same, the protocol has changed as described in Section 1.3.1 (see page 9).

### Core Pattern at SOC Level

R	D	A	A	A	A	R	D	W	
E	A	D	D	D	D	D	E	A	R
F	T	D	D	D	D	D	S	T	S
C	A	R	R	R	R	R	E	A	T
L	I	[0]	[1]	[2]	[3]	[4]	T	O	N
K	N							U	
	[0]							T	
								[0]	
1	0	0	0	0	0	0	1	x	0
1	0	0	0	0	1	0	1	x	0
1	1	0	0	1	1	0	0	x	1
1	1	0	0	1	1	0	0	x	1
1	0	0	0	1	1	0	0	x	1
1	0	0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0	0	1
1	0	0	0	1	1	0	0	0	1

Figure 5 Core Pattern at SOC Level

### 2.1.2 Partition-Based Testing, Debug and Diagnosis

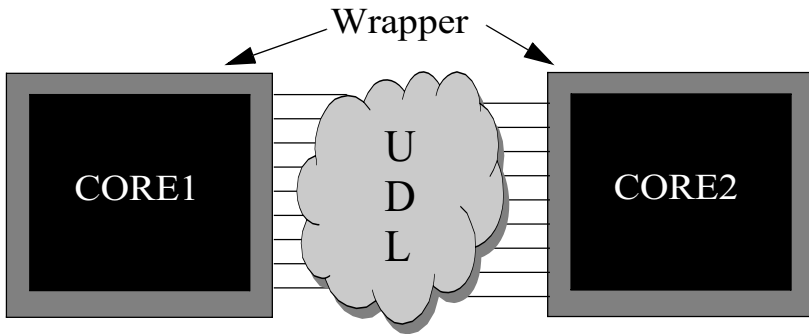
Test, debug and diagnosis of an SOC can be an overwhelming task. If the SOC is partitioned, this task becomes much more manageable. If a pattern fails on the tester and the SOC is not partitioned, it takes much work to figure out which part of the SOC caused that failure. On the other hand, if the SOC has been partitioned into wrapped cores and a pattern fails, the area of the failure can more easily be traced to a specific entity and no other wrapped core

needs to be addressed during debug and diagnosis. Power during test can also be measured per core if the power domains are segmented properly. The wrappers also partition the logic external to the cores, so that if there is any area of issue in the external logic, it can also be found more easily.

## 2.2 Why Was a Standard Needed?

There are many advantages to standardization. There are no surprises in the implementation of the standard as tools can be created and the flow automated, literature is written to explain how best to use a standard and even improve on it.

Currently cores from multiple companies or even from different divisions within a company, can be utilized in an SOC. The cores could be a combination of hard and soft cores. A hard core is one whose library has been chosen, layout and routing is complete and all test structures are in place; in fact it is ready to go into a specific fabrication process. Hard cores users need only instantiate the hard core into an SOC of the same technology. A soft core is one that is delivered in behavioral language. The customer must do all timing, layout, routing and much of the test structure. However, the customer has the freedom to choose the technology, test methodology, etc. of this core. These cores may or may not have wrappers. In addition, there may be user-defined logic (UDL) external to the cores that must also be tested as shown in Figure 6 (page 19). The SOC designer must determine how to connect these entities up for functional use. In addition, the SOC designer must determine how to test all of the logic in the SOC. Each of the hard cores may have a different test methodology, which the SOC designer must understand and make work together. If the cores do have wrappers, it may be difficult to get core wrappers synchronized in order to test the UDL properly. The soft cores may need their test methodology developed by the SOC designer, as does the UDL. This is a lot of extra work to assign to the SOC designer and it may be a high risk flow due to the fact that the SOC designer may not be familiar with the cores. This flow may also take an excessive amount of time. If the interface for each of these cores was the same or similar and that interface allowed for testability of user-defined logic, this would make the SOC designers job much easier, more timely and less prone to mistakes. If this interface is standardized, it would allow for ease of automation of wrapper creation, manipulation of patterns, SOC test connections and test scheduling.



**Figure 6** Simultaneous Access of Two Wrappers

The 1500 standard describes a standardized interface that will allow for all of the advantages listed above. This standard is rigid in some areas that require two or more cores' wrappers to be utilized in conjunction with each other. However, it is quite relaxed in other areas, such as the types of tests that can be allowed for testing cores or UDL. The standard lists a plethora of example wrapper cells to meet most test requirements. There is a standard mechanism for delivering instructions to all cores, a standard mechanism for bypassing wrapper boundary registers (WBRs) and a standard pin interface for use on the wrapper instruction register (WIR), wrapper bypass register (WBY) and WBR. It is a great combination of rules and flexibility that should allow automation of the creation of the wrappers, the stitching of test signals in an SOC and in conjunction with CTL, pattern manipulation for embedded cores and test scheduling.

The Core Test Wrapper Handbook  
Rationale and Application of IEEE Std. 1500™  
da Silva, F.; McLaurin, T.; Waayers, T.  
2006, XXIX, 276 p., Hardcover  
ISBN: 978-0-387-30751-0