Structured DFT Development Approach for Chisel-Based High Performance RISC-V Processors

1st Bin Zhang	2nd Ye Cai*	3rd Zhiheng He	4th Sen Liang	5th Wei He
ShenZhen University	*ShenZhen University*	*Beijing Institute of open source chip*	*Institute of Computing Chinese Academy of Sciences*	*PengCheng Laboratory*
ShenZhen, China	ShenZhen, China	BeiJing, China	BeiJing, China	ShenZhen, China
2060271073@email.szu.edu.cn	Caiye@szu.edu.cn	hezhiheng@bosc.ac.cn	liangsen20z@ict.ac.cn	hew@pcl.ac.cn

***Abstract*—Research has shown that agile language Chisel and related agile design methodology is promising to sustain the scaling computing performance in a more efficient way. Design For Test, as an economical and effective method for chip production testing, must be integrated into the agile development system to meet the requirements of mass production. Due to the lack of support for the traditional EDA toolchain, chip development through Chisel has not been widely accepted in the industry. This paper is based on the research of XiangShan, an open-source project for RISC-V high-performance processor. XiangShan establishes a structured DFT(degisn for test) development approach and Chisel-based DFT agile design flow. XiangShan develops a flexible chisel-based XS-shared bus mbist interface to improve design PPA and proposes the MarchSLD algorithm to enhance defect detection in the FinFet process node. XiangShan's DFT development Approach is moving towards hierarchical, flexible, and reliable in two generations of processors. Experimental results show that the optimized Chisel-based DFT design flow can support agile development requirements and achieve industry-competitive performance.**

***Index Terms*—Chisel, Design For Test, Shared bus MBIST, March Algorithm**

I. Introduction

Research has shown that agile language Chisel and related agile design methodology is promising to sustain the scaling computing performance in a more efficient way. Chisel is a hardware programming language for agile development proposed by a research team in Berkeley, California, in 2012[1]. Chisel is derived from the Scala programming language, inherits the high-level features of Scala, and can reduce code size, increase code readability, and improve development efficiency compared to traditional Verilog.

A. Chisel's key features

The main feature of Chisel is its abstraction from high-level programming languages, which is reflected in the following advantages for hardware programming[2]:

- Simplifies the complexity of signal connections
- Supports metaprogramming
- Use the object-oriented nature of high-level languages
- Supports functional programming

B. Chip Agile Design and DFT

The relevant team has built open-source chip agile development platforms around the Chisel-based design[3]. However, there are still many issues that need to be addressed before agile chip design can move towards practical applications. In order to effectively and economically complete the production testing of chips, design for test(DFT) is essential. Thus, how to integrate DFT into agile chip development is also a sub topic.

There are mature solutions in the industry for embedding structured DFT into RTL or gate-level netlist, but research on Chisel-based DFT has not been widely carried out. On high-performance processors, Intel has maintained product evolution through the Tick-Tock startegy. With constantly design and technology scaling, it may lead to high costs of outdated DFT. To address these challenges, this paper proposes an optimized Chisel-based DFT design flow that can support agile development while also achieving industry-competitive performance.

II. Related Work

The primary decision to use the Chisel language for agile development was made for "XiangShan", an open-source, high-performance RISC-V processor. "XiangShan" has been widely influenced by the international open-source community and has evolved over three generations of microarchitecture. YanQiHu-NanHu-KunMingHu, providing an excellent open-source platform for industry and academia.

A. Agile Development of Chisel-based Open-source Chips

In [2] Yu Z H et al. extensively discussed the agile language Chisel and related agile development methodologies. The conclusions led to the creation of the open-source, high-performance processor, the XiangShan project. In [3], Xu Y et al. proposed the MINJIE development platform to support an agile design flow for developing complex high-performance processors. They successfully applied the MINJIE platform to the XiangShan processor.

The first-generation XiangShan processor with the "YanQiHu" microarchitecture, clocked at 1GHz @ 28nm, was sample back and tested in January 2022 and achieved SPEC

CPU2006 7@1GHz with DDR4-1600 in the SPEC score, which was highly consistent with the expected performance evaluation. The successful back-to-socket test of the XiangShan processor represents the entire feasibility of Chisel-based agile development of open-source chips. The second generation of XiangShan processors with the "NanHu" microarchitecture and DFT included will be ready for silicon in Q1 2023.

B. DFT Industry Practices For High Performance Processors

ARM is proposing a new MBIST Interface in [4], called shared bus, that effectively optimizes the impact of the MBIST Controller on the chip's power, performance and area(PPA). ARM uses this technology on high-performance cores.

In [5], the shared bus learning flow is presented, which implements physical-to-logical (P2L) automation. The P2L mapping automation can improve development efficiency while reducing the probability of making errors.

In [6], it is mentioned that technology scaling, design scaling, and system scaling are all challenging for DFT. Especially in design scaling, Tessent proposes the shift left DFT, true hierarchical DFT, and streaming scan network, which are great inspirations for DFT design solutions of XiangShan processors.

In advanced process nodes, DFT needs to capture more manufacturing defects in the fabricated wafers to maintain a low level of DPPM. March algorithm of MBIST is constantly updated, and the algorithm's complexity is increasing; ATPG introduces more fault models, cell-aware, time-aware, etc. In [7], Said Hamdioui et al. studied the fault behavior of connectivity faults and proposed the March SL algorithm, which achieves 100 percent fault coverage for connectivity faults. However, the detection capability for dynamic faults is weak.

III. Towards Hierarchical Flexible and Reliable DFT Solution in XiangShan

A. Building a Base DFT Design Flow in YanQiHu

Chisel, as a new high-level hardware description language, has not overturn the traditional toolchain ecosystem. Chisel-based design adds layers to the chip development toolchain. It supports compiling into RTL for compatibility with traditional chip development flows, including synthesis, DFT design, verification based on the UVM framework, etc. Fig. 1 show that Chisel-based design can support DFT RTL flow and DFT Gate flow. DFT RTL flow is recommended to take advantage of physical synthesis and agile verification.

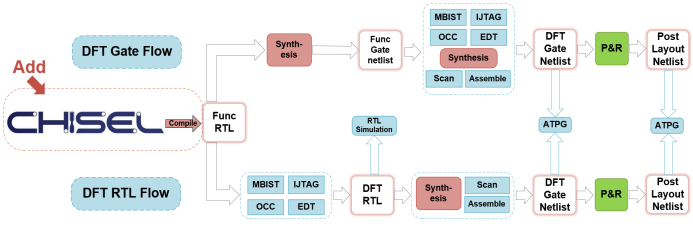

Fig. 1. DFT Design Flow

We have completed the complete process design and verification of DFT on YanQiHu. DFT design data is shown in session IV Experimental results. Chisel-based design does not have significant limitations on DFT development and can also achieve high test coverage. We begin to further consider whether DFT can be more agile and optimize the PPA of chips if we complete some DFT development on Chisel. Referring to DFT industry practices for high performance processors, we attempt to shift DFT more left in Chisel.

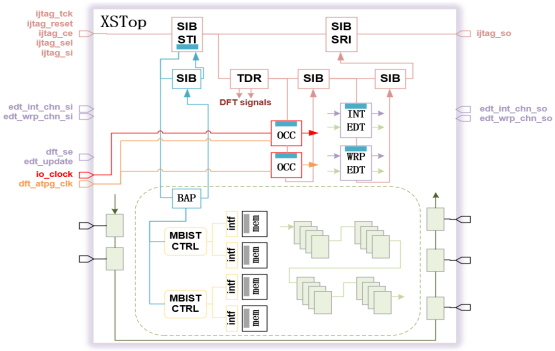

Fig. 2. YanQiHu DFT Design Architecture

B. Flexibale Chisel-based XS-SharedBus Interface in NanHu

Compared to YanQiHu, NanHu introduces L3 Cache with a larger cachesize, and the clock frequency is increased to 2GHz@14nm. In the NanHu development, we proposed the XS-shared bus architecture to optimize the MBIST circuit and moved the development work forward in the Chisel.

In the design of the XS-shared bus, we decouple it from the functional modules of the CPU, and the XS-shared bus only acts as an IP module of the CPU. Its existence or not will not affect the internal functional logic of the CPU. Thanks to the high-level features of the chisel language, we can directly exemplify the shared bus circuit module wherever it is needed, and its interfaces are all of the already implemented Bundle types, so there is no need to do additional interface wiring operations, thus realizing plug-and-play.

Mbistarray in XS-shared bus is automatically scheduled, pipelined memory read and write access; the pipeline stage that reuses functions as much as possible. It also supports testing power consumption mode and read and write latency.

Cause of we can transmit the specifications of the memory to the MBIST interface via variables and parameters, XS-shared bus circuit can be self-adapted when the size and number of memories are changed. To control the chip area, the maximum width of mbistdata is set to 256. Logical memory with data width greater than 256 will be splited out into multiple nodes. When the size of logical memory changes, the number and size of nodes are recalculated based on the size of the parameter.

The code in the Fig. 3 shows how to calculate a way for multiple nodes. The first line of code uses the language features of the Chisel to find all the common factors of the width of a way in just one line.

We have also implemented a flexible pipeline placement scheme, allowing different pipeline topologies to be designed according to requirements. We represent both ram and pipeline

```
①val divisors=Seq.tabulate(end)(_+1).map(idx=>(in%idx==0, Seq(idx,in/idx))).filter(_._1).flatMap(_._2).sorted
②val validDivisors = divisors.filter(_<=mw)
③val goodNodeNumForEachWay = dw / validDivisors.max
④val defaultNodeNumForEachWay = ((dw + mw - 1) / mw)
⑤val finalNodeNumForEachWay = if (goodNodeNumForEachWay > 4 * defaultNodeNumForEachWay)
    defaultNodeNumForEachWay else goodNodeNumForEachWay
    (finalNodeNumForEachWay, way * finalNodeNumForEachWay)
```

Fig. 3. The code example of XS-shared bus

as node, with each node including a level attribute. The level of the ram node is constant at 0, and the level of the pipeline is customizable. When instantiating a node, it compares the level value with the existing nodes in the global, and if there is a node with a level value less than the current node, it will be added to the children of the current node.

The Fig. 4 shows that since the module is instantiated depth-first, a different topology can be achieved by defining different level parameters when instantiating the second pipeline.

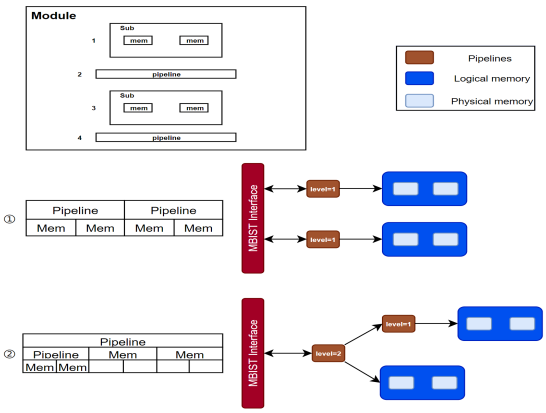

Fig. 4. The example of pipeline placement and topology

When using a third-party EDA tool to integrate the MBIST controller based on the shared bus approach, we must let the tool know the interface-to-memory mapping relationship. ARM communicates mapping information to third-party EDAs via MBIF files[4]. Without MBIF, it's a troublesome job for custom-shared bus designs.

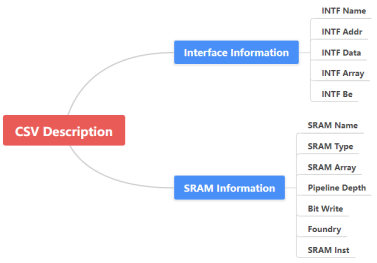

Fig. 5. The Basic Structure of CSV

In XS-shared bus, We can automatically generate the mapping information due to the advantage of Chisel parameterization passing. We use the PrintWriter function in the Chisel code to describe the above information in a CSV file as the Fig. 5 shown, which can be opened directly in text or through Excel to visualize the mapping relationships.

The code in the Fig. 6 represents how the interface information is exported to a csv file in Chisel.

```
①val fileHandle = new PrintWriter(f"build/$infoName.csv")
②val intfHeads = "\"INTF Name\", \"INTF Addr\", \"INTF Data\",\"INTF Array\", \"INTF Be\"\n"
③fileHandle.print(intfHeads)
④fileHandle.print(intfInfo.toString + "\n")
```

Fig. 6. The code example of CSV generation

In addition, we provide conversion scripts to convert csv files into MBIST inputs for mainstream EDA tools, together with EDA's own sharebus learning flow, to finally achieve shared bus DFT flow automation in the Fig. 7.

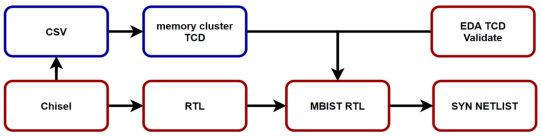

Fig. 7. Nanhu MBIST Flow

From the XS-SharedBus MBIST, chisel's agile development process significantly reduces development costs compared to the traditional industrial DFT process. It takes only 1-2 people familiar with chisel development and a DFT designer to complete the work from code development to functional verification in a week's time. The shared bus mbist circuit developed with chisel also has the advantages of adaptive adjustment and automatic numbering and grouping.

C. March SLD Algorithm in NanHu

As the size and number of memories increase, the probability of dynamic faults increases. Dynamic faults are divided into single-cell dynamic defects and dynamic coupling faults.

The March SL and SLE algorithm can adequately cover connection and static faults but is weak in detecting dynamic defects. The March SLE algorithm can fully cover the dynamic faults of dRDF, dDRDF, dIRF, dCFrd, dCFdrd, and dCFir, but it cannot fully cover some cases of dWDF, dCFwd and dCFds. The table I lists the faults that are not detected in them, and their corresponding fault primitive.

TABLE I
FAULT TYPES THAT MARCH SLE CANNOT COVER

FFM	FP
dWDF	0w0w0/↑/-, 1w1w1/↓/-
dCFwd	0;0w0w0/↑/-, 1;0w0w0/↑/-, 0;1w1w1/↓/-, 1;1w1w1/↓/-
dCFds	0w0w0;0/↑/-, 1w1w1;0/↑/-, 0w0w0;1/↓/-, 1w1w1;1/↓/-

We have changed each element of the March algorithm without affecting the SLE algorithm and its original detection sequence, adding coverage for a subset of faults dWDF, dCFwd,

and dCFds. The Fig. 8 represents the specific sequence of operations for the March SLD algorithm, with the new operations added for the dWDF and dCFwd faults in red font. And in blue font are the operations added for the dCFds fault.

```
{⇕(w0);
M0;
⇑(r0,w0,w0,r0,r0,w1,r1,w1,w1,r1,r1,w0,r0,w0,r0,w1);
M1;
⇑(r1,w1,w1,r1,r1,w0,r0,w0,w0,r0,r0,w1,r1,w1,r1,w0);
M2;
⇓(r0,w0,w0,r0,r0,w1,r1,w1,w1,r1,r1,w0,r0,w0,r0,w1);
M3;
⇓(r1,w1,w1,r1,r1,w0,r0,w0,w0,r0,r0,w1,r1,w1,r1,w0); }
M4;
```

Fig. 8. March SLD

Firstly, two write operations are added to the beginning of the element. This allows the algorithm to cover both dWDF and dCFwd faults. For the dCFds fault, the successive write operations at the beginning do not obtain the fault value immediately, so it is necessary to add subsequent write operations to sensitize the defective cell inside each element. And a read operation is added at the beginning of the next stage to read the fault value of the affected faulty cell. The table II describes the phases in which the SLD algorithm is able to detect faults.

TABLE II
MARCH SLD FAULT DETECTION PHASE

FFM	FP	Phase
dWDF	0w0w0/↑/-	M1,3 M2,9 M3,3 M4,9
dWDF	1w1w1/↓/-	M1,9 M2,3 M3,9 M4,3
dCFwd	0;0w0w0/↑/-	M1,3 M2,9 M3,3 M4,9
dCFwd	1;0w0w0/↑/-	M1,9 M2,3 M3,9 M4,3
dCFwd	0;1w1w1/↓/-	M1,3 M2,9 M3,3 M4,9
dCFwd	1;1w1w1/↓/-	M1,9 M2,3 M3,9 M4,3
dCFds	0w0w0;0/↑/-	M1,0 M3,0
dCFds	1w1w1;0/↑/-	M2,0 M4,0
dCFds	0w0w0;1/↓/-	M1,0 M3,0
dCFds	1w1w1;1/↓/-	M2,0 M4,0

The March SLD algorithm can integrate the detection of single-unit and two-unit dynamic faults in a single detection algorithm compared to existing algorithms for dynamic faults such as March MD[8] and reduces the complexity of the algorithm compared to March MD2[8].

To verify the effectiveness of the algorithm, a generic and portable fault simulation method is proposed. As shown in the Fig. 9, the read/write signals, data, and address signals passed in the interface module are synchronously input to the fault simulation module, while the data contents read in the memory are obtained. The fault simulation module makes a judgment of the fault simulation condition at each rising edge of the clock. If the fault triggering condition is met, the value is processed and output to the comparison module.

We have demonstrated the capability of the March SLD algorithm for dynamic fault coverage using dWDF(0w0w0/↑/-), dCFwd(0;0w0w0/↑/-) and dCFds(0w0w0;0/↑/-) in the Fig. 10.

D. Hierarchical DFT Integrated Optimization in Nanhu

NanHu performs a series of integrated optimization compared with YanQiHu. One is shared bus MBIST and March

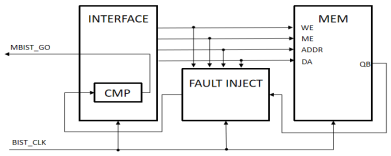

Fig. 9. Fault Simulation Module

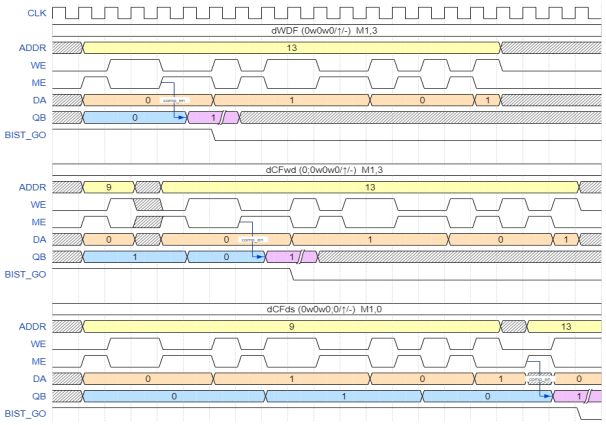

Fig. 10. March SLD Fault Detection

SLD Algorithm integration mentioned above. To compensate for the negative impact of shared bus MBIST in test time, NanHu performs the integration of Streaming Scan Network(SSN), a new generation SCAN integration solution in industry. The DFT circuit architecture of NanHu is shown in the Fig. 11.

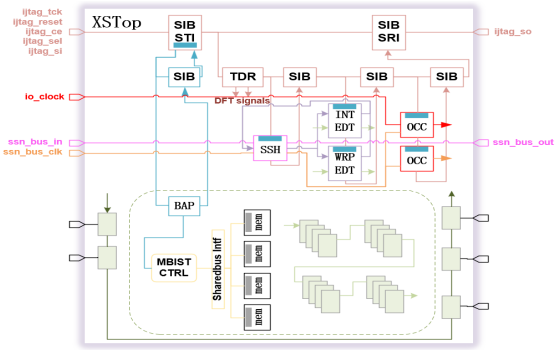

Fig. 11. NanHu DFT Circuit Architecture

On-chip compare makes constant test costs in indentical cores, which is better to support the multi-core scaling of NanHu. In [9], intel presents benefits of SSN for testing of complex SoCs. In the integration of SSN, EDT is internally integrated via Streaming Scan Hosts(SSH), adjusting EDT compression ratio is no longer affecting the module port. we call it true hierarchical design.

In addition, due to the huge L3 cache size, we have also added MBISR for Memory Repair.

The new DFT Design Flow is shown in the Fig. 12,more and more DFT IPs will be developed in Chisel further.

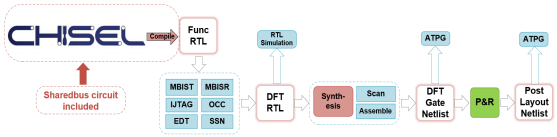

Fig. 12. The New Dft Design Flow

IV. Experimental Results

A. Hierarchical DFT Practice in XiangShan

Xiangshan is an open-source project,we have also released open-source DFT reference flow for YanQiHu and NanHu in [10, 11]. In reference examples, the DFT RTL IP is generated through the Tessent platform, and synthesis and scan stitching is performed in Design Complier(DC) platform. ATPG is also performed by Tessent ATPG. The table III shows the specific design data comparison between Nanhu and YanQiHu.

TABLE III
Design Data Comparison

YanQiHu	dual core NanHu
16KB L1 ICache 128KB L1plus Icache 32KB L1 Dcache 1MB inclusive L2 Cache	128KB L1 ICache*2 128KB L1 Dcache*2 1MB non-inclusive L2 Cache*2 6MB non-inclusive L3 Cache
1.3GHz@28nm	2GHz@14nm
17 Mbist Controller	8 Mbist Controller total 2 Mbist Cntr per XSTile
MBIST algorithm complex 44N	MBIST algorithm complex 68N
	Extra March SLD Hard-Coded L3 Cache MBISR
MBIST test time 4.37ms	MBIST test time 17.86ms
3.86M Instances	7.9M Instances total 2.6M Instances per XSTile
508k ScanReg	1.59M ScanReg total 658K ScanReg per XSTile
Scan pin Total 25 Global Signal:3 Int edt channel 10:10 Wrp edt channel 1:1	Scan pin Total 33 Ssn_bus_clk:1 Ssn_bus_in/out 16:16
Scan shift@48MHz	Scan shift@100MHz ssn_bus_clk@200MHz
Stuck-at cov 99.92% Transition cov 98.20%	Stuck-at cov 99.89% Transition cov 97.86%
SA Pattern count 22660 Tr Pattern count 41313	SA Pattern count 24119 Tr Pattern count 45559
SA+TR test time 434.41ms	SA+TR test time 281.4ms

DFT area growth ratio of YanQiHu and NanHu is shown in the Fig. 13. Shared bus logic is not included in NanHu area data before DFT. IJTAG, MBIST, OCC, EDT, SCAN replacement, SCAN stitching is included in area data after DFT. In addition, ssn and mbisr is included in NanHu area data after DFT.

The comparison data mentioned above, we can see:

- YanQiHu and NanHu XSTile have similar structure and Instances size. DFT area growth optimized from 21.3% to 12.7%. The overall area is reduced by 46.6% cause of 14nm process node
- XSTop's DFT area growth ratio is only 3.4% in huge cache size. Shared bus is more suitable for dealing with these scenarios to reduce mbist controllers and friendly place and route

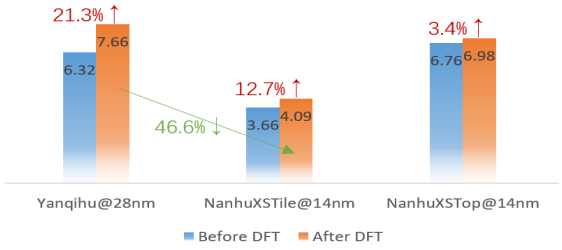

Fig. 13. Comparison of the area growth of YanQiHu and NanHu

- Memory serial testing leads to increased testing time in sharedbus. NanHu optimizes ATPG testing time by increasing shift frequency and upgrading SCAN integration architecture to SSN
- In the comparsion of test time for basic test items, which is composed by mbist, stuck-at and transition, NanHu test time is only 68.2% of YanQiHu. Although instances of NanHu is more than 2x of YanQiHu

In addition, the MarchSLD algorithm is integrated into the design as the Extra user-define algorithm. Within the scope of static, connectivity, and dynamic faults studied in this paper, the coverage, complexity, and circuit synthesis data of the SLD algorithm are compared with several other algorithms in the table. The March SLD algorithm provides complete coverage of the types of faults studied. Although the algorithm's complexity is increased, it does not severely impact the circuit and is within the acceptable range.

TABLE IV
Algorithm Comparison

Algorithm	Coverage	Complexity	Area	Power
March SL	79.14%	41N	14723814	2.35E+03
March SLE	97.0%	49N	14724015	2.35E+03
March SLD	100%	65N	14724430	2.35E+03

B. Recommended SCAN integration solution for many-core XiangShan processors

Many-core XiangShan processor has achived better optimization due to the integration of SSN. The many-core XiangShan processor consists of 8 clusters, with a dual core Nanhu microarchitecture as a cluster. The chip has 65 GPIOs that can be reused by SCAN, with maximum IO freq. up to 100MHz.

1) test time: On-chip compare makes constant test costs for indentical cores design, table V shows the testing content of many-core XiangShan processor.

TABLE V
Test Content Comparison

	Test Group1	Test Group2
SSN ATPG Test	**Retargetable Pattern:** XSTop(I)+XSTile(E) **Retargeting Group:** CLUSTER0-CLUSTER7	**Retargetable Pattern:** XSTile(I) **Retargeting Group:** CLUSTER0-CLUSTER7

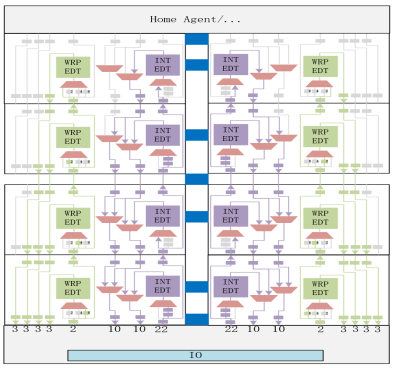

Fig. 14. Pin-muxed integration of the many Core processor.

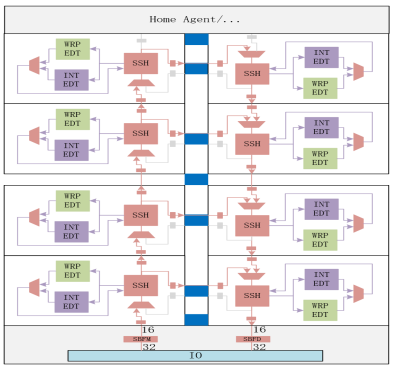

Fig. 15. SSN integration of the many Core processor.

Test time statistics are as follows in the table VI. The testing time of 283.4ms for 8 cluster is only slightly more than the testing time of 281.4ms for 1 cluster. Because pipeline is added into scan integration of many-core processor for timing closure.

TABLE VI
TEST TIME STATISTICS

	SA_GP1	TR_GP1	SA_GP2	TR_GP2	TOTAL
SSN	22.1ms	62.0ms	77.2ms	122.1ms	283.4ms

2) integration: From the comparison of integration types between Pinmuxed and SSN in the Fig. 14 and Fig. 15, it can be seen that the test parallelism of Pinmuxed is limited by IO resources, while SSN is not limitd and is suitable for design scaling; SSN realizes less signal interaction between design modules, and the serial connection between design modules is more suitable for tile-based design.

V. Conclusion

Stuck-at 99.5+% and transiton 95+% test coverage in two generations of XiangShan processor, chisel-base design can also achieve high test coverage. Furthermore, XiangShan improves the coverage of SRAM dynamic fault detetion to 100% by the March SLD algorithm. We will further collect the effectiveness of the March SLD algorithm on DPPM during chip testing. Not only does the Chisel-based design not set obstacles for the evolution of structured DFT, but the flexible Chisel-based XS-shared bus interface makes DFT development more agile. Base on hierarchical DFT integrated optimization in NanHu, the DFT area growth ratio is optimized from 21.3% to 12.7%. The test time of basic test items is only 68.2% of YanQiHu, although instances of NanHu is more than 2x of YanQiHu. Many-core XiangShan processor has achieved better optimization both in test time and integration used by SSN. In summary, facing the scaling of Chisel-based high-performance RISC-V processors, we proposed the optimized Chisel-based DFT design flow that can support agile development requirements and achieve industry-competitive performance.

Acknowledgment

I sincerely thank my postgraduate supervisor, Mr. Cai, Mr. He, and all my colleagues and classmates for their hard work and help. This work was funded in part by Shenzhen Fundamental Research Program 20230723121846002 and SZUGS2023JG06.

References

[1] Bachrach J, Vo H, Richards B, et al. Chisel: constructing hardware in a scala embedded language[C]//Proceedings of the 49th Annual Design Automation Conference. 2012: 1216-1225.

[2] Yu Z H, Liu Z G, Li Y W. Practice of chip agile development: Labeled RISC-V[J]. Journal of Computer Research and Development, 2019, 56(1): 35-48.

[3] Xu Y, Yu Z, Tang D, et al. Towards Developing High Performance RISC-V Processors Using Agile Methodology[C]//2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2022: 1178-1199.

[4] McLaurin T, Knoth R. The Challenges of Implementing an MBIST Interface: A Practical Application[C]//2019 IEEE International Test Conference (ITC). IEEE, 2019: 1-6.

[5] Harshitha Kodali. Automation of shared bus memory test with Tessent MemoryBIST[M]. United States: Siemens Digital INndustries Software, 2022.

[6] Ron Press. Manage scaling challenges for silicon success[M]. United States: Siemens Digital INndustries Software, 2021.

[7] Hamdioui S, Al-Ars Z, Van De Goor A J, et al. Linked faults in random access memories: concept, fault models, test algorithms, and industrial results[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2004, 23(5): 737-757.

[8] Harutunyan G, Vardanian V A, Zorian Y. Minimal March tests for detection of dynamic faults in random access memories[J]. Journal of Electronic Testing, 2007, 23: 55-74.

[9] Côté J F, Kassab M, Janiszewski W, et al. Streaming scan network (SSN): An efficient packetized data network for testing of complex SoCs[C]//2020 IEEE International Test Conference (ITC). IEEE, 2020: 1-10.

[10] YanQiHu DFT flow https://zhuanlan.zhihu.com/p/482312961

[11] NanHu DFT flow https://zhuanlan.zhihu.com/p/591264508