Integrating Core Selection in the SOC Test Solution Design-Flow

Erik Larsson

Embedded Systems Laboratory
Linköpings Universitet
SE-582 83 Linköping
Sweden
erila@ida.liu.se

Abstract[1]

We propose a technique to integrate core selection in the SOC (system-on-chip) test solution design-flow. It can, in contrast to previous approaches, be used in the early design-space exploration phase (the core selection process) to evaluate the impact on the system's final test solution imposed by different design decisions, i.e. the core selection and the cores' test characteristics. The proposed technique includes the interdependent problems: test scheduling, TAM (test access mechanism) design, test set selection and test resource floor-planning, and it minimizes a weighted cost-function based on test time and TAM routing cost while considering test conflicts and test power limitations. An advantage with the technique is the novel three-level power model: system, power-grid, and core. We have implemented and compared the proposed technique, a fast estimation technique and a computational extensive pseudo-exhaustive method, and the results demonstrate that our technique produces high quality solutions at reasonable computational cost.

1 Introduction

In a core-based design environment, cores (pre-designed and pre-verified blocks of logic and UDL (user-defined logic) blocks) are integrated by the *core integrator* to become a system, which can be placed on a single die, making it an SOC (System-on-Chip). The cores, provided by *core vendors*, may each have different origin, such as from various companies, reuse from previous designs, or the cores can be completely new in-house designs. The *core test integrator* is responsible for the design of the system's test solution, which includes decision on how and when test data (test stimuli and test response) should be transported and applied to each core in the system.

A SOC design flow is usually as follows. First, a design step (core selection step) where the core integrator selects the appropriate cores for the system, and second, as a consecutive step, the core test integrator designs the test solution for the system, which includes test scheduling and the design of the infrastructure for test data transportation, the TAM (Test Access Mechanism).

It is important to note that, the core integrator can, in the initial design step (core selection), select among a number different cores often from several core vendors to implement a certain functionality in the system. The core integrator selects, based on each core's design characteristics, the cores that fits the system best. Each possible core may, not only have different design characteristics, but can also have different test characteristics (for instance test sets and power consumption). For example, one core may require a large ATE (Automatic Test Equipment) stored test set, while another core, implementing the same functionality, requires a combination of a limited ATE test set and a BIST (Built-In Self-Test) test set. Note, that even if a single core is available for a function, the partitioning (ATE size versus BISTs size) is to be decided. Selecting the optimal core for a possible functionality leads to local optimum, however, not necessarily to a global optimum. In other words, the selection of cores must be considered with a system perspective in order to find a globally optimized solution. It means that there is need for a test solution design tool that can be used in the early core selection process to explore and optimize the system's test solution.

We have previously proposed a technique for integrated test scheduling and TAM design where a weighted cost-function based on test time and TAM wiring cost is minimized while considering test conflicts and test power consumption [11]. We have also previously proposed an estimation-based technique for test set selection and test resource placement [10]. In this paper we propose an integrated technique including *test set selection*, *test resource floor-planning*, *TAM design,* and *test scheduling.* The test set selection, test resource floor-planning, TAM design, and test scheduling are highly inter-dependent. The test time can be minimized by scheduling the tests as concurrent as possible, however, the possibility of concurrent testing depends on the size of the TAM connecting the test resources (test sources and test sinks). The placement of the test resources has a direct impact on the length of the TAM wires. And finally, the selected test sets for each testable unit are partitioned over the test resources and impacts the TAM design and the test schedule.

1. The research is partially supported by the Swedish National Program STRINGENT.

The proposed technique includes an improved power model that consider global system-level limitations, local limitations on power-grid level (hot spots) as well as core-level limitations. The motivation for a more elaborate power model is that the system is designed to operate in normal mode, however, during testing mode the testable units are activated in a way that would not occur during normal operation. It can lead to (1) that the systems power budget is exceeded, or (2) that hot spots appear and damage a certain part in the system, or (3) that a core is activated in such a way that the core is damaged.

We have implemented the proposed technique, the estimation-based technique [10] as well as a pseudo-exhaustive technique, and through experiments we demonstrate that our proposed technique produces high quality solutions at low computational cost.

The rest of the paper is organized as follows. A background and an overview of prior work is in Section 2. The problem formulation is in Section 3, and the test problems and their modeling are in Section 4. The algorithm is described in Section 5, and an example illustrating the algorithm is in Section 6. The experimental results are in Section 7, and the conclusions are in Section 8.

2 Background and Related Work

A core-based design flow a sequence that typically starts with core selection, followed by test solution design, and after production, the system is tested (Figure 1(a)). In the core selection stage, the core integrator selects appropriate cores to implement the intended functionality of the system. For each function there is usually a number of possible cores to select from, and where each candidate core has its specification on, for instance, performance, power consumption, area, and test characteristics. The core integrator explores the design space (search and combines cores) in order to optimize the SOC. Once the system is fixed (cores are selected) the core test integrator designs the TAM and schedules the tests based on the test specification for each core. In such a design flow (illustrated in Figure 1(a)), the test solution design is a consecutive step to core selection. And, even if each core's specification is highly optimized, when integrated as a system, the system's global test solution is most likely not highly optimized.

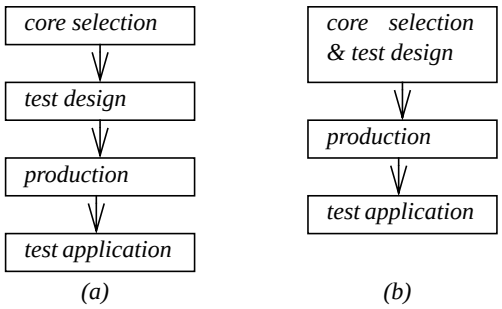

Figure 1. Design flow in a core-based design environment (a) traditional and (b) proposed.

The design flow in Figure 1(b), on the other hand, integrates the core selection step with the test solution design step, making it possible to consider the impact of core selection when designing the test solution. In such a design flow (Figure 1(b)), the global system impact on core selection is considered, and the advantage is that it is possible to develop a more optimized test solution. The design flow in Figure 1(b) can be viewed as in Figure 2, where the core type is floor-planned in the system but there is not yet a design decision on which core to select. For each position, several cores are possible. For instance, for the cpu_x core there are in Figure 2 three alternative processor cores (cpu1, cpu2 and cpu3).

In this paper we make use of the definition of concepts introduced by Zorian *et al.* [15], which we illustrate with an example in Figure 3. The example consists of three main blocks of logic, core A (CPU core), core B (DSP core), and core C (UDL (user-defined logic) block). A test source is where test stimuli is created or stored, and a test sink is where the test response from a test is stored or analyzed. The test resources (test source and test sink) can be placed on-chip or off-chip. In Figure 3 the ATE serves as an off-chip test source and off-chip test sink, while TS1, for instance, is an on-chip test source. The TAM is the infrastructure (1) for test stimuli transportation from a test source to the testable unit, and (2) for test response transportation from a testable unit to a test sink. A wrapper is the interface between a core and the TAM, and a core with a wrapper is *wrapped* while a core without wrapper is *unwrapped*. Core A is a wrapped core while Core C is unwrapped. The wrapper cells at each wrapper can be in one of the following modes at a time: *internal mode*, *external mode* and *normal operation mode*.

In addition to the definitions by Zorian *et al*. [15], we assume that a testable unit is not a core, but a block at a core and that a core can consist of a set of blocks. For example, core A (Figure 3) consists of two blocks (A.1 and A.2).

For a fixed system where cores are selected and floor-planned, the main task is to organize the testing and the transportation of test stimuli and test response (as the example design in Figure 3). Several techniques have been proposed to solve different important problems under the

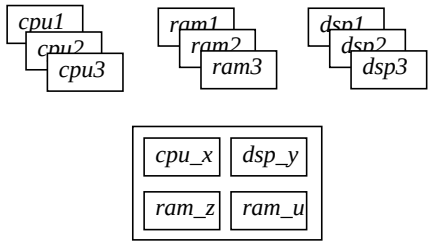

Figure 2. System design.

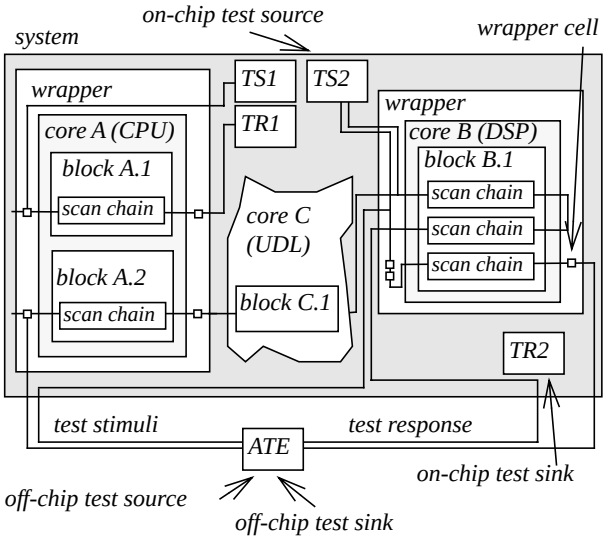

Figure 3. A system and the definition of test concepts.

assumption that the cores are already selected (design flow as in Figure 1(a)).

Zorian [14] proposed a test scheduling technique for fully BISTed system where each testable unit is tested by one test with a fixed test time, and each testable unit has its dedicated test source and its dedicated test sink. A fixed test power value is attached to each test and the aim is to organize the tests into sessions in such a way that the summation of the power consumed in a session is not more than the system's power budget while minimizing the test application time.

In a system where each testable unit has its dedicated test source and test sink, there are no test conflicts. Chou *et al*. proposed a test scheduling technique minimizing the test time for systems where both the test time and power consumption for each test are fixed, and to handle general conflicts a conflict graph is used [1].

The approaches by Zorian and Chou *et al.* assume fixed testing times for each testable unit. However, the test time for scan-tested cores is often not fixed. If the scanned elements (scan-chains, inputs, and outputs) at a core are connected into few wrapper chains, the testing time is higher compared to if the scan elements are connected into a higher number of wrapper chains. Iyengar *et al*. proposed a scheduling technique for systems where the testing time for all cores is flexible and the objective is to form a set of wrapper chains for each core in such a way that the testing time for the system is minimized [7].

The test power for a test can also be fixed or flexible. Bonhomme *et al*. [2] and Saxena *et al.* [12] proposed clock-gating schemes intended to reduce the test power consumed during the scan-shift process. The advantage is that the test power can be reduced at a core with such a scheme. If a wrapper chain consists of *n* scan-chains, the scan-chains can be loaded one at a time, which means that only one chain is active at a time, hence, lower power consumption.

Sugihara *et al.* investigated the partitioning of test sets where one part is on-chip test (BIST) and the other part is off-chip test using an ATE (Automatic Test Equipment) [13]. A similar approach was proposed by Jervan *et al*. [8], which later was extended to not only locally optimize the test set for a core but to consider the complete system by using an estimation technique to reduce the test generation complexity [9]. Hetherington *et al*. discussed several important test limitations such as ATE bandwidth and memory limitations [5].

All the above addressed problems are important. But it also important to consider them all from a system test perspective.

We have previously proposed an integrated technique for test scheduling and TAM design where the test application time and the TAM design are minimized while considering test conflicts and power consumption [11]. And, we have proposed an estimation technique for the creation and optimization of the TRS (*test resource specification*) [10]. The objective was to create a TRS that together with the design specification will be the inputs to a test scheduling and TAM design tool and hence result in an efficient test solution, *i.e.* minimal test application time and minimal routing of the TAM wires.

3 Problem Formulation

An example of an input specification, the starting point in our approach, is given in Figure 4. The structure of the input specification is based on the one we used in [11] with one major extension, namely that for each block several lists of tests can be specified, instead of as before where it was only possible to assign one list per block. The advantage with the approach is that it is possible to specify several lists of tests for each block (testable unit) where each test in a list make use of its specified resources (test source and test sink), and each test has its test characteristics. The test problems that are considered in our technique and their modeling is discussed in Section 4. The cores are floor-planned, *i.e*. given (x,y) coordinates and each core consist of a set of blocks (testable unit). For each block, several sets of tests are available, where each set of tests is sufficient for the testing the block. For instance, to test a block bA three possible test sets are given:

```
[Blocks] name idle pwr pwr_grid {test1, t2,..., tn} {t1,..tn}
         bA    0   grid1 {tA1, tA2} {tB1} {tC1 tC2 tC3}
```

{tA1, tA2} or {tB1} or {tC1 tC2 tC3} should be selected where each test has its resources and characteristics.

A way to view the problem is shown in Figure 5. A set of test resources (test sources and test sinks) are available. Different combinations of sets of test sets for each testable unit can be defined. If CoreA is selected {t1, t2} or {t3} can be selected. The tests make use of test sources and test sinks and the selection of other cores impact on the total resource

```
[Global Options]
MaxPower = 100
[Power Grid] #name     power_limit
             p_grid1     50
             p_grid2     60
[Cores]   #name   x    y    block_list
          coreA   20   10   { blockA1, blockA2 }
          coreB   40   10   { blockB1, blockB2 }
          coreC              {  blockC1 }
[Generators] #name    x    y    max_bw memory
             ATE      10   0    4          200
             TG1      30   0    1          50
             // the rest of the generators
             TG2      30   10   1          100
[Evaluators] #name    x    y    max_bw
             ATE      50   0    4
             TRE1     30   0    1
             // the rest of the evaluators
             TRE2     30   10   1
[Tests]   #name pwr  time tpg   tre    min_bw  max_bw  ict
          tA1.1 60   60   TG1   TE1    1       1       no
          // more tests for coreA
          tB1.1 60   72   TG1   TE1    1       1       no
          // more tests for coreB
          tC1.1 70   80   TG1   TE2    1       4       coreB
          // more tests for coreC
[Blocks] #name   idle_pwr pwr_grid   test_sets {}, {}, ...,{}
          blkA1     0      p_grd1    { tA1.1 }{ tA1.2, tA1.3}
          blkA2     0      p_grd1    { tA2.1 }{ tA2.2 }
          blkB1     5      p_grd2    { tB1.1 tB1.2 } { tB1.3 }
          blkB2     10     p_grd2    { tB2.1 }
          blkC1     0      p_grd1    { tC1.1}
[Constraint] #name  {block1, block2, ... , blockn}
          tA1.1     {}
          // constraints for each test
          tC1.1     {blkC1 blkA1 blkA2 blkB1 blkB2}
```

Figure 4. Input specification for the example system in Figure 3.

usage, and hence, the total cost. The problem is to select, for each block, which set of tests to use in order to produce an optimized test solution for the system. The cost of a test solution is given by the test application time and the amount of routed TAM wires:

$$cost = \alpha \times \tau_{total} + \beta \times TAM \qquad 1$$

where τ_{total} is the test time (end time of the test with highest test time), *TAM* is the routing length of all TAM wires, and, α and β are user-defined constants used to determine the importance of test time in relation to TAM cost.

The produced output from our technique is (1) a test schedule where the tests are (2) selected, (3) given a start time, and (4) an end time in such a way that all conflicts and constraints are not violated, and (5) a corresponding TAM layout where the cost (Eq. 1) is minimized.

4 Test Problems and Modeling

In this section we discuss the test problems to be considered when developing a SOC test solution and their modelling.

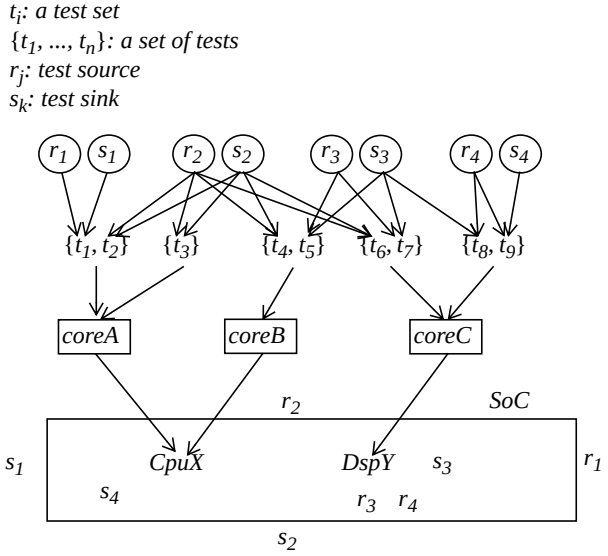

Figure 5. Illustration of design alternatives.

4.1 Test Time

The testing time for a testable unit can be *fixed* or *non-fixed* prior to the design of the test solution. A core provider can optimize the core and its wrapper making the testing time fixed. On the other hand, the testing time for a scan-tested core is often non-fixed since the scan-chains can be connected into one or more wrapper-chains. The testing time for a test with flexible test time depends on the number of wrapper-chains. Important to note is that tests with fixed and non-fixed testing times can be mixed in the system.

A core can consist of scan-chains that are few and unbalanced (of unequal length), and the testing time might not be linearly dependent on the number of wrapper chains. Therefore, we have analyzed if the testing time (τ) is linear with the number of wrapper-chains (w) ($\tau \times w = constant$) for the scan-tested cores in one of the largest ITC'02 designs, namely the P93791 design. We observed that the testing time for core 11 was most non-linear (Core 11 - original in Figure 6). We noted that the 576 scanned elements were partitioned into 11 scan-chains (82 82 82 81 81 81 18 18 17 17 17). We re-designed core 11 into four new cores with 11, 22, 44, and 88 balanced scan-chains, respectively. We plotted $\tau \times w$ for all cores in Figure 6. As the number of scan-chains increases, the more constant the value $\tau \times w$ becomes. The testing time at a single wrapper chain is 149381 (marked in Figure 6). For core 11 with 44 balanced scan-chains, the value $\tau \times w$ is always less than 5% from the constant theoretical value. Important to note is that for all cores, the value $\tau \times w$ is almost constant within a certain range. We assume that core test designers optimize the cores, hence, the number of scan-chains at a core is relatively high and of nearly equal length.

In our model, we specify the testing time for a testable

unit at a single TAM wire and the bandwidth limitations. For instance a testA has a test time of 100 at a single wrapper chain and the scanned elements can be in the range 1 to 4:

```
[Tests] name  test time    minbw    maxbw
        testA  100          1        4
```

and we assume, based on the experiments, that the test time is linear to the number of TAM wires within the bandwidth range. It means that given the test time at a single TAM wire, the test time can be computed:

$$\tau_i = \frac{\tau_1}{tam} \qquad 2$$

where *tam* is in the range [*minbw,maxbw*]. If the testing time is fixed, *minbw=maxbw.*

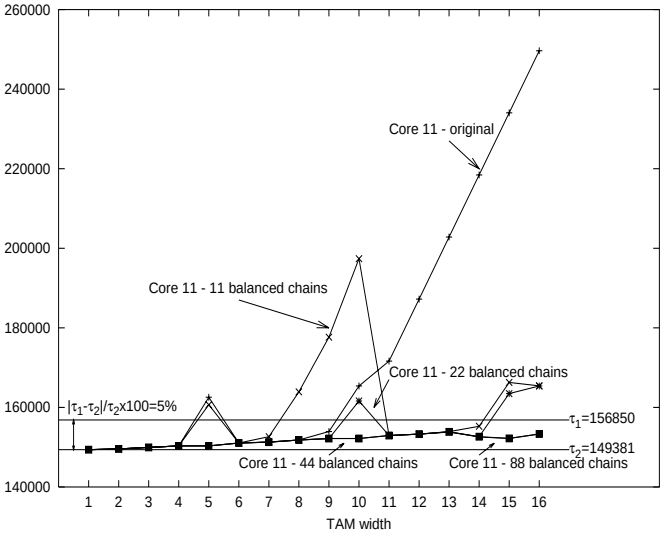

Figure 6. Test time analysis for core 11 in design P93791.

4.2 Test Power Consumption

The power dissipation during testing mode is often higher than that during normal operation. The reason is that power consumption is highly related to the switching activity, and in order to detect as many faults as possible at a minimum of time, it is desirable to activate (switch) as many fault locations as possible in a short period of time. Furthermore, the system is designed to operate during normal operation, but during testing mode blocks are made active in a way that would not occur in normal mode. It means that special care has to be taken to the power dissipation during testing.

The testing time is reduced if a high number of cores are activated concurrently, but it leads to higher activity, and high power consumption can damage the system. The system-level power budget can be exceeded in such a scheme. Furthermore, if cores that are physically close are activated during testing, a hot spot can be created and the system is damaged. For instance, assume a memory organized as a bank of four where in normal operation only one bank is active at a time. However, during testing, in order to shorten the test time, all banks are activated at the same time. The system's power limit might not be exceeded, however, a local hot spot is created, and the system is damaged. Another problem is that a core during testing mode dissipates power above its specified limit due to the nature of the test stimuli and/or the test clock frequency. We therefore make use of a three-level power model: *system-level*, *power-grid-level (local hot spot)*, and *core-level*.

For the system-level, we make use of the power model defined by Chou *et al*. where a fixed power value is attached to each test, and the tests are scheduled in such a way that at any time point, the summation of power values executed concurrently is below the power budget of the system [1].

As an example, we specify the system budget:

MaxPower = 100

and for each test we specify the power consumed when the test is activated:

```
[Tests] name pwr time tpg tre minbw   maxbw mem ict
        testA 60  60   r1  s1     1        1     10    no
```

the idle power, the power consumed when a block is not active is specified at each block:

```
[Blocks] name idle pwr pwr_grid {test1, t2,..., tn} {t1,..tn}
         bA    0       grid1    {testA} {testA2}
```

For local hot spots, we introduce a power grid model, which has similarities to the approach proposed by Chou *et al*. [1], but in contrast to it, our model include local areas (power grids). We assume that each block (testable unit) is assigned to a power grid where the power grid has its power budget. The system can contain a number of power grids. Blocks assigned to a power grid cannot be activated in such a way that the power grid budget is exceeded at any time.

An example to illustrate the need of power grids is as follows, a memory can be organized as a bank of memory blocks (see Figure 7). Assume that the memory, during normal operation, never accesses more than a single memory block the power grid is designed accordingly. .

As an example of a single grid is:

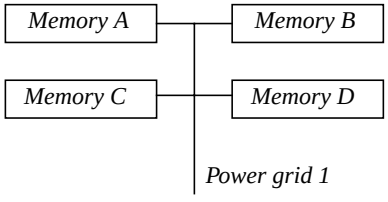

Figure 7. A memory organized as a bank of four blocks powered by a common grid.

```
[PowerGrid]   pwr_grid        limit
              grid1           30
```

and for each block the used power grid is given:

```
[Blocks] name idle_pwr pwr_grid {test1, t2,..., tn} {t1,..tn}
         bA   0        grid1    {testA} {testA2}
```

The motivation behind core-level adjustments is two-fold. First, by lowering the power consumption at a core, a higher number of cores can be activated concurrently without violating the system power budget. Second, since test power consumption often is higher than that during the normal operation, the power dissipation during test at a specific core can be higher that its own power budget, which can damage the core.

As discussed above, some tests have a fixed testing time while other tests allow flexible testing times. Regarding test power consumption, we have some tests where the power is fixed regardless of the number of assigned TAM wires, while other tests allow the power to be adjusted. The power can be adjusted by using clock-gating [12]. Clock-gating can be used to reduce the power consumption so that a higher number of tests can be executed concurrently, but more importantly, it can be use for testable units where its own power dissipation is higher than its allowed power consumption due to for instance a too high test clock frequency.

The power consumption for a test is given as a single value, for instance as in the following example:

```
[Tests] name pwr time minbw maxbw flexible_pwr
        testA 50  60   1     4     yes
        testB 60  30   1     4     no
```

Note that we include the possibility to specify if clock-gating can be used by setting *flexible_pwr* to yes or no. If power can be modified, we assume a linear dependency:

$$p_i = p_1 \times tam \qquad 3$$

where p_1 is the power at a single tam wire, and *tam* is the number of TAM wires, which has to be in the specified range [*minbw*:*maxbw*].

4.3 Test Conflicts

During the test solution design there are a number of conflicts that have to be considered and modelled. For general conflicts we make use of the following notation:

```
[Constraints] test{block1, block2, ..., block n}
              tA   {bA bB}
```

It means that when testing tA both block bA and bB must be available since they are used by test tA or tA might interfere with one of the blocks. This modelling support general conflicts, which can be given from hierarchy where cores are embedded in cores or from interference during testing.

A test source ([Generators]) may have limited bandwidth and memory. We model bandwidth limitation as an integer stating the highest allowed bandwidth for the test source. For memory limitations an integer is used as the upper memory limit. A test sink ([Evaluators]) can also have a limited bandwidth and in a similar way as with test sources, we model it as an integer. For each test we give a number of its memory requirement. An example with testA using test source r1 and test sink s1:

```
[Generators] name   x   y   maxbw   memory
             r1     10  20  1       100
[Evaluators] name   x   y   maxbw
             s1     20  20  2
[Tests]      name   tg      tre     mem
             testA   r1     s1      10
```

The wrapper conflicts are slightly different compared to general conflicts because of the TAM routing. The testing of a wrapped core is different from the testing of an unwrapped. The testing of the wrapped core A (Figure 3) is performed by placing the wrapper in *internal test mode* and test stimuli is transported from the required test source using a set of TAM wires to the core and the test response is transported from the core using a set of TAM wires to the test sink. In the case of an unwrapped testable unit such as the UDL block, the wrappers at core A and B are placed in *external test mode*. The test stimuli is transported from the required test source on the TAM via core A to the UDL block and the test response is transported via core B to the TAM and to the test sink.

We model the wrapper conflict as in the following example with two blocks (bA and bB) and one test per block (tA and tB):

```
[Blocks] name  {test1, test2,..., test m} {test1, ..., test n}
         bA    {tA}
         bB    {tB}
[Tests] name   tg    tre    ict
        tA     r1    s1     bB
        tB     r1    s1     no
```

Test tB is not an interconnection test, hence, itc is marked as no. It means that there will be a connection between r1 to bB and from bB to s1. Test tA, on the other hand, is an interconnection test with bB. It means that r1 is connected to bA and bB is connected to s1.

4.4 Resource Utilization

We make use of a machine-oriented Gantt chart to track bottlenecks (the resource that limits the solution) [3]. We let the resources be the machines, and the tests the jobs to show the allocation of jobs on machines. For example, a Gantt chart as in Figure 8 where for instance, test B2 needs TG:r2

```
1. Select tests for initial solution
2. Do {
3.    Create test schedule and TAM
4.    Find limiting resource with Gantt chart
5.    Modify tests (select alternative tests or modify
      TAM width) at limiting resource
6.    Select best modification
7. } Until no improvement is found
8. Return best solution.
```

Figure 9. The algorithm.

and TRE:s2. An inspection of Figure 8 shows that TG:r2 and TRE:s2 are not limiting the solution. On the other hand, test source TG:r1 is the most critical one. It means that testA, testB, and testC are the obvious candidates for modification. The Gantt chart pin-points bottle-necks and therefore reduces the search for candidates for modification. Note that the Gantt chart does not show a schedule, only the usage of resources in the system.

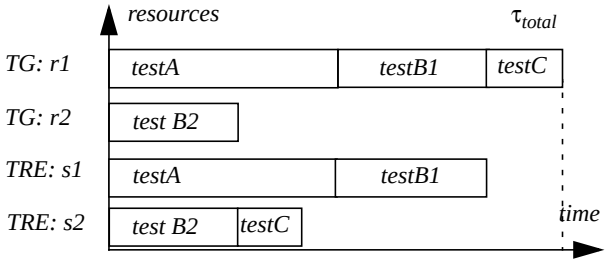

Figure 8. A machine-oriented Gantt chart [3].

5 Algorithm

In this section we describe the proposed algorithm (outlined in Figure 9, and detailed in Figure 10 and Figure 11).

In order to evaluate the cost of a test solution, we make use of (Eq. 1). At a design modifications, the cost change before and after modification, is given by:

$$= \Delta\tau \times \alpha + \Delta TAM \times \beta \qquad 4$$

where $\Delta\tau$ (ΔTAM) is the difference in test time (TAM cost), before and after the modification.

The *TAM* cost is given by the length l and its width w ($TAM=l \times w$), and by combining the cost function (Eq. 1) considering only one testable unit, and the test time versus TAM cost (Eq. 2), the optimal TAM bandwidth is given by [11]:

$$w = \sqrt{(\alpha \times \tau)/(\beta \times l)} \qquad 5$$

A detailed description of the algorithm (Figure 9) is in Figure 10 (test set selection outline) and Figure 11 (test scheduling and TAM design). The algorithm starts by the part given in Figure 10, where the list of test sets for each testable unit is sorted based on the cost function (Eq. 1). The

```
1. sort the list of tests descending according to the cost function.
2. repeat until the list is empty {
3.    select the first test in the list
4.    repeat until a test is scheduled or at end of list {
5.       repeat until selected test is scheduled or
         bandwidth cannot be decreased {
6.          try to schedule the test at current time
7.          if fail to schedule {
8.             if the bandwidth>1 then reduce bandwidth with 1
9.          }
10.      }
11.      if the selected test could not be scheduled {
12.         select the following test in the list
13.      }
14.   }
15.   if the test was scheduled {
16.      allocate TAM and remove the selected from the list
17.   } else {
18.         update current time to the nearest time in the future
            where there is available power to schedule the first test
            in the list
19.      }
20. }
```

Figure 11. Test scheduling and TAM design algorithm.

cost for each testable unit is locally optimized, however, there is at this point no global consideration on sharing of TAM wires or conflicts. For each testable unit, the first set of tests for each testable unit is selected, and the set is scheduled and the TAM is designed (Figure 11). From the test schedule, the test application time is given and from the TAM layout, the TAM cost for the solution is given. The algorithm checks the use of resources from a Gantt-chart for the solution. For example, assume the a test solution generates a Gantt-chart as in Figure 8, where TG:r1 is the critical resource. For all tests that are using the critical (limiting) resource, we try to find alternative tests. We make use of Eq. 4 to evaluate the change in cost for each possible alternative (at the critical resource). Instead of trying all possible alternatives, we try a limited number of design modifications. And to reduce the TAM cost we try to make use of existing TAMs (a test can be delayed and applied later). We make use of Eq. 4 where the difference in test time and TAM cost are given.

6 Example

We use the example design in Figure 12 to illustrate the algorithm in Section 5. The example (Figure 12), simplified by removing power grids, memory limitations and the list of general constraints, consists of two cores each with a single block (testable unit). Each block can be tested in two ways; there are two alternative test sets for each block. For instance, blockA can be tested by testA1 or by testA2.

The algorithm proceeds as follows. Initial step: For each block, the test sets are ordered ascending according to the cost function (Eq. 1 assuming $\alpha=\beta=1$):

```
1. for each block (testable unit) {
2.    for each test set at a block {
3.       compute optimal bandwidth for each test (Eq. 5);
4.       compute cost for the full test set (Eq. 1);
5.    }
6.    place the test sets sorted descending on the cost (step 4.);
7.    select the first test set in the list as the active test set
8. }
9. repeat until no modification can be performed {
10.   create test schedule and TAM layout (see Figure 11)
11.   if the total cost for schedule and TAM layout is best so far{
12.      remember this test schedule and the TAM layout
13.   }else {
14.      if last modification was a TAM width modification {
15.         undo the TAM width modification
16.      }
17.      if last modification was a test set modification {
18.         remove the test set from the blocks list of test sets
19.      }
20.   }
21.   for each block {
22.      if the active test set has a test resource limiting
         the solution {
23.         compute cost for increasing the TAM width with 1
24.         for every other test set for the block{
25.            compute the cost of changing this test set
               based on Eq. 4
26.            if the cost is lower than lowest cost {
                     remember this test set
27.            }
28.      }
29.      if lowest cost for the block is lower than the total cost{
30.         remember block, TAM width and test set modification
31.      }
32.   }
33.   if any alternatives exists {
34.      perform TAM width modification or
         test set modification
35.   } else {
36.      for each block {
37.         for each test set after the active test set for the block{
38.            compute the cost of selecting it (Eq. 4)
39.            if cost is lowest then remember this test set
40.         }
41.         if lowest cost for the block is lower than lowest total
               cost then {
42.               remember block change and test set change
43.         }
44.      }
45.      if lowest cost difference <0 {
46.         do the test set change
47.      }
48.   }
49. }
```

Figure 10. Test set selection algorithm.

test	time	TAM	total cost
testA1:	60	40	100
testA2:	100	20	120
testB1:	72	40	112
testB2:	120	20	140

The evaluation results in the following sorted lists per block (first in the list is the best candidate):

BlockA: {{testA1}, {testA2}}
BlockB: {{testB1}, {testB2}}

The first set of tests are selected as active, that is for BlockA {testA1} and for BlockB {testB1}. The test scheduling algorithm sorts the tests based on test time, and starts with the longest test, making the test schedule: testB starting at time 0 followed by testA starting at time 72. The resulting total test application time is 132. The TAM design algorithm connects TG1, coreB, coreA, and TA1, and the Manhattan length is 20+20+20=60. The total cost (at $\alpha=\beta=1$) for the test solution is then: 132 (test time)+60 (TAM cost)=192.

From the Gantt chart for this test solution, we observe that TG1 and TA1 both are used for 132 time units, while TG2 and TA2 are not used at all, and we note that TG1 and TA1 limits the solution. Based on the Gantt-chart, the algorithm tries to find an alternative that is not using TG1 and TA1. For each test that uses the limiting resources in the Gantt chart, in our example TG1 and TA1, the algorithm computes the alternative cost of using other resources. It is important to note, that in order to limit the number of possible options, we only try with the tests that are the resources that limits the solution (Gantt chart).

First alternative modification, we try using testA2 to test BlockA instead of using testA1. It means that testA1 will not be executed (one of the set of tests for each block is only required). We evaluate the impact of the test modification on the TAM layout, and we observe that we do not have to include coreA in the layout. Taking coreA out of the bus layout means that TAM corresponding to 20 units can be removed (testA2 is making use of different test resources compared to testA1). However, in order to execute testA2 we have to include wires from TG2 to coreA and from coreA to TA2. The additional required wiring corresponds to 20 units.

The difference in test time between testA1 and testA2 is (100-60=) 40. It means that the total cost difference is estimated to: -20 (what we gain by not including coreA for testA1)+20 (what we have to add to include TAM for TG2->coreA->TA2)+40=40.

Second alternative modification, we try testB2 instead of testB1. It means that a TAM (length and width) corresponding to 20 units can be removed. The additional TAM cost of adding testB2 (its resources) is 20, and the difference in test time between testB2 and testB1 is 48 (120-72). The cost difference for this alternative is -20+20+48=48.

In this example we have two tests using the resources that limits the solution (Gantt chart), and we also had only one possible alternative per test. And, since the first alternative is better than the second (40 is less than 48), the first is

selected. A new test schedule and a TAM layout is created where both testA1 and testB1 are scheduled to start at time 0, and there are two TAMs, one from TG2->coreA->TA2 at length 20, and one from TG1->coreB->TA1 at length 40. The total cost is 60+72=132 (an improvement from 192 to 132).

7 Experimental Results

The objective with the experiments is to check that the proposed technique produces high quality solutions at reasonable computational cost (CPU time). We have therefore implemented the proposed technique (described above), the estimation-based technique proposed by Larsson and Fujiwara [10] and a pseudo-exhaustive technique. The motivation for having an estimation-based technique is to demonstrate that finding a high quality test solution is not trivial, and the idea of having a pseudo-exhaustive technique, that basically tries all possible solutions, is to demonstrate that the search space is enormous in size.

We have created three designs, which in increasing size are: *design_small*, *design_medium* and *design_large*. Design_small contains 4 cores each with one testable unit and for each testable unit there are two alternative tests. Design_medium contains 13 cores also with one testable unit per core and for each testable unit there are 5 alternative tests. Design_large consists of 122 testable units distributed over 18 cores and 186 tests.

```
[Global Options]
MaxPower = 100
[Cores]      #name    x    y    block_list
             coreA    20   10   { blockA }
             coreB    40   10   { blockB }
[Generators] #name    x    y    max_bw
             TG1      30   0    1
             TG2      30   10   1
[Evaluators] #name    x    y    max_bw
             TA1      30   0    1
             TA2      30   10   1
[Tests]      #name pwr time TPG TRE  min_bw  max_bw ict
             testA1 60   60   TG1 TA1  1       1      no
             testB1 60   72   TG1 TA1  1       1      no
             testA2 40   100  TG1 TA1  1       1      no
             testB2 40   120  TG1 TA1  1       1      no
[Blocks]     #name   idle_power   test_sets
             bA      0            { testA1 }{ testA2 }
             bB      0            { testB1 }{ testB2 }
```

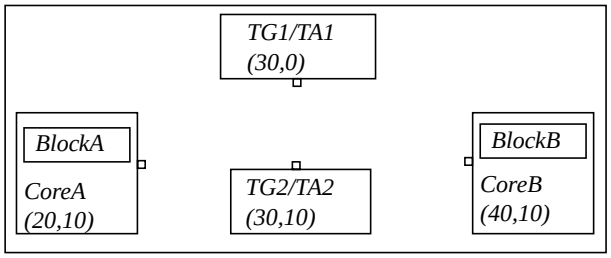

Figure 12. An illustrative example with a simplified input where power grids, memory limitations, and constraint list (general constraints) are not considered.

The experimental results using the three techniques on the three designs are collected in Table 1, Table 2, Table 3, and Table 4. Table 1 reports the computational cost (CPU time) for each of the three techniques. The estimation-based technique produces results very quickly (less than one second) while the pseudo-exhaustive approach is terminated for the two larger designs, *i.e.* no results were produced within reasonable time. The proposed technique used CPU time that is in between the estimation-based and the pseudo-exhaustive approach, and results were produced for all designs at acceptable CPU times. For the largest design the computational cost was 4 seconds.

For the quality of the solutions we have collected the test application time, the TAM cost, and the total cost of the test solution for each of the three techniques at each of the three designs (reported in Table 2, Table 3, Table 4, respectively). The test application time for design_small of the solution produced by the estimation-based technique is 400, for the solution produced by both the pseudo-exhaustive technique and the proposed technique the test time is 320 (Table 2). The solution from the estimation-based technique is 25% worse than the solutions from the pseudo-exhaustive and the proposed techniques. The experiment indicates that the proposed technique finds a solution of high quality (the same test time as the pseudo-exhaustive technique).

The TAM cost for the three techniques at the three designs are collected in Table 3. The proposed technique finds for design_small a test solution with the same TAM cost (120) as the pseudo-exhaustive technique. The results from estimation-based technique is 140, which 17% from the pseudo-exhaustive and the proposed techniques.

The total cost is computed with $\alpha=\beta=1$ making the total cost = test time + TAM cost (Eq. 1). For instance, the total cost for the proposed technique at design_small is 440 and it comes from the summation of the test time 320 (Table 2) and the TAM cost 120 (Table 3). The proposed technique produces a solution at the same cost as the pseudo exhaustive technique for design_small, while the solution from the estimation-based technique is 23% worse. The proposed technique produces results for all three designs that are better than the results from the estimation-based technique.

Design	Estimation [10]	Pseudo-exhaustive	Proposed technique
Design_small	<1	<1	<1
Design_medium	<1	N.A	<1
Design_large	<1	N.A	4

Table 1. Computational cost (seconds).

Design	Estimation [10]	Pseudo-exhaustive	Proposed technique
Design_small	400	320	320
Design_medium	240	N.A	193
Design_large	215	N.A	220

Table 2. Test application time.

Design	Estimation [10]	Pseudo-exhaustive	Proposed technique
Design_small	140	120	120
Design_medium	810	N.A	690
Design_large	1072	N.A	962

Table 3. TAM routing cost.

Design	Estimation [10]	Pseudo-exhaustive	Proposed technique
Design_small	540	440	440
Design_medium	1050	N.A	883
Design_large	1287	N.A	1182

Table 4. Total cost (α=β=1).

8 Conclusions

In the paper we have proposed a technique where the design of the test solution is included as early as in the core selection phase. The advantage with the technique is that it makes it possible to explore the impact of different core alternatives when searching for a final test solution. In contrast to prior work, our approach, does not only include test scheduling and TAM design, but also test set selection and test resource placement, which are highly dependent on the test schedule and the TAM layout.

Our technique can, for instance, be used to explore the placement of test resources (test sources and test sinks) in the system. Also, the technique can be used to explore the partitioning between tests can be explored (the ATE size versus the BIST size). And finally, the approach makes it possible to explore the impact of different cores and their different test characteristics (test sets, test resource use, test time, test power etc.) on the final test solution. The search space is large for a technique that include test scheduling, TAM design, test set selection, and test resource placement. In order to limit the search in the design space, we make use of Gantt charts to find limiting resources (bottle-necks).

We have also improved the test power to a three level power budget model; system-level, power-grid (local hot-spot) level, and clock-gating at core-level. The advantage is higher control on where the power is consumed.

For validation of the technique, we have implemented and compared the proposed technique, an estimation-based technique and a pseudo-exhaustive technique. The experimental results indicate that our approach produces good results at reasonable computational cost.

References

[1] R. M. Chou, K. K. Saluja and V. D. Agrawal, “Scheduling Tests for VLSI Systems Under Power Constraints”, *Transactions on VLSI Systems*, Vol. 5, No. 2, pp. 175-185, June 1997.

[2] Y. Bonhomme, P. Girard, L. Guiller, C. Landrault, and S. Pravossoudovitch, “A Gated Clock Scheme for Low Power Scan Testing of Logic ICs or Embedded Cores”, *Proceedings of Asian Test Symposium (ATS)*, pp. 253-258, November 2001.

[3] P. Brucker, “Scheduling Algorithms”, *Springer-Verlag*, ISBN 3-540-64105-X, 1998.

[4] A. L. Crouch, “*Design For Test*”, Prentice Hall PTR, 1999.

[5] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski, “Logic BIST for Large Industrial Designs: Real Issues and Case Studies”, *Proceedings of International Test Conference (ITC),* pp. 358-367, September 1999.

[6] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, “Test Access Mechanism Optimization, Test Scheduling, and Tester Data Volume Reduction for System-on-Chip”, *Transactions on Computers*, December 2003 (Vol. 52, No. 12), pp. 1619-1632.

[7] V. Iyengar, K. Chakrabarty, and E. J. Marinissen, “Co-Optimization of Test Wrapper and Test Access Architecture for Embedded Cores”, *Journal of Electronic Testing; Theory and Applications (JETTA),* pp 213-230, April 2002.

[8] G. Jervan, Z. Peng, R. Ubar, and H. Kruus, “A Hybrid BIST Architecture and its Optimization for SoC Testing”, *Proceedings of International Symposium on Quality Electronic Design (ISQED'02)*, pp. 273-279, March 2002.

[9] G. Jervan, P. Eles, Z. Peng, R. Ubar, and M. Jenihhin, “Test Time Minimization for Hybrid BIST of Core-Based Systems”, *Asian Test Symposium (ATS’03)*, Xian, China, November 17-19, 2003, pp. 318-323.

[10] E. Larsson and H. Fujiwara, “Test Resource Partitioning and Optimization for SOC Designs”, *Proceedings of VLSI Test Symposium (VTS’03)*, Napa, USA, 27 April - 1 May 2003, pp. 319-324.

[11] E. Larsson, K. Arvidsson, H. Fujiwara, and Z. Peng, “Efficient Test Solutions for Core-based Designs”, *Transactions on CAD of Integrated Circuits and Systems*, pp. 758-775, May 2004.

[12] J. Saxena, K. M. Butler, and L. Whetsel, “An Analysis of Power Reduction Techniques in Scan Testing”, *Proc. of International Test Conference (ITC)*, pp. 670-677, Oct. 2001.

[13] M. Sugihara, H. Date, and H. Yasuura, “Analysis and Minimization of Test Time in a Combined BIST and External Test Approach”, *Proceedings of Design and Test in Europe (DATE),* pp. 134-140, March 2000.

[14] Y. Zorian, “A distributed BIST control scheme for complex VLSI devices”, *Proceedings of VLSI Test Symposium (VTS)*, pp. 4-9, April 1993.

[15] Y. Zorian, E. J. Marinissen, and S. Dey, S., “Testing Embedded-Core Based System Chips”, *Proceedings of International Test Conference (ITC),* 1998, pp. 130 - 143, October 1998.