As of March 2011, TRACE32 trace information can be streamed to a host hard-disk from the running target. The large amount of program flow data which can result from this method, leads to a significant simplification of the code-coverage.

Trace-based Code-Coverage

Proof of statement coverage and condition coverage is often required to meet system quality specifications in industries such as medical and automotive.

- **Statement coverage** proves that each line of code was executed during the system test.
- **Condition coverage** proves that for each conditional instruction both pass and fail branches were executed at least once.

For many embedded systems highly optimized code must be tested in real-time. The alternatives of code instrumentation and non-real-time operation cannot be used in these cases.

To be able to meet these requirements, the target processor/SoC must fulfill the following prerequisites:

1. The cores which are implemented must have a core trace logic (see figure 1). This logic generates information about the instructions executed by the core. Depending on the operation of the trace logic, information about the task switches and the read/write operations can also appear.

2. The processor/SoC must have a trace port with sufficient bandwidth so that the trace information can be recorded by an external tool without any information loss.

The Classic Measurement Process

Until now, code-coverage analysis was performed with TRACE32 using the following steps:

1. Start program execution and automatically stop when the trace memory is full.
2. Transfer the trace memory content to the code-coverage database.
3. Continue program execution.

For each measurement step, the amount of data collected was limited by the size of the memory available within the trace tool. The results of the code-coverage-analysis could be checked after the total measurement was completed or, if needed, after each intermediate step.

New: Streaming

If the trace data is transferred to a drive on the host computer at the time of recording, the complete software routine can be recorded in one measurement step. The streamed data is stored within a file on the hard-disk. To avoid completely filling the hard-disk with trace data, TRACE32 stops streaming as soon as less than 1 GByte of free memory remains.

To be able to stream, the following technical prerequisites must be fulfilled:

- 64-bit host computer and 64-bit TRACE32 executable
- Interface between trace tool and host computer must be as fast as possible.
- Optimal configuration of the trace source and the trace tool

![Diagram of the code-coverage analysis process](image)

Fig. 1: For the code-coverage analysis, up to 1 TByte of trace data can be streamed to the host computer.
Fast Host Interface

The amount of trace data that is exported via the trace port depends on the target system hardware. The number of cores, the number of trace port pins, and the trace clock speed are all important parameters. The protocol used by the core trace logic plays also an important role. For example, the ARM PTM protocol is more compact than the ARM ETMv3 protocol (see figure 2).

The embedded software is another major variable. A software program that performs many jumps and retrieves data/instructions mainly from the cache produces more trace data per second than a software program that processes many sequential instructions and must frequently wait for the availability of data/instructions.

The amount of data varies but it is always large. Streaming only works properly, if the transfer rate between the tool and the host computer is fast enough to transfer all of the data from the trace port to the host computer without any data loss. The 1 GBit Ethernet interface is the only recommended interface for the PowerTrace II.

The programming of the trace logic on the chip can be used to directly influence the amount of trace data being generated. The logic should be programmed so that only trace information which is relevant to the code-coverage analysis is being generated. To illustrate this point, the following two examples are provided.

ETM/PTM: Optimal Configuration

ETM and PTM are different implementations of the core trace logic on the ARM/Cortex architectures. The ETM can be configured so that trace information is produced only for the instructions executed by the program. Information about the read/write operations is not needed for code-coverage. By default the PTM only generates information about the program flow. Therefore the PTM does not need to be configured.

PowerTrace vs. PowerTrace II

TRACE32 trace tools are available in two designs, which differ especially in relation to their features.

**PowerTrace**
- 256 or 512 MByte trace memory
- USB 2.x and 100 MBit Ethernet
- 80 MBit/s as maximum transfer rate to host computer
- Software compression of trace data (factor 3)
- Memory interface with 100 MHz

**PowerTrace II**
- 1/2/4 GByte trace memory
- USB 2.x and 1 GBit Ethernet
- 500 MBit/s as maximum transfer rate to host computer
- Hardware compression of trace data for ETMv3 and PTM (factor 6)
- Memory interface with 233 MHz

---

**Fig. 2:** A transmission rate of 3.2 GBit/s is generally adequate for streaming program sequence information on the host.
Both trace sources encode the virtual address instructions. If an embedded design uses an operating system, such as Linux or Embedded Windows, virtual addresses cannot be mapped unambiguously to physical addresses. The trace source must also be configured, so that information is generated defining the virtual address space in which an instruction was located.

For the ARM ETM/PTM, the amount of trace data can be further reduced:

- The code-coverage analysis does not analyze or need time information. We therefore recommend configuring the TRACE32 trace tool so that the trace data is transferred to the host without time stamps. This reduces the amount of data by a third.
- PowerTrace II also provides FPGA-based hardware compression of the trace data. This enables up to 3.2 GBit/s trace data to be transferred to the host computer. Figure 9 shows that this transfer rate is generally sufficient for streaming ETM/PTM data without any data loss.

**Nexus: Optimal Configuration**

On processors of the MPC5xxx/SPC5xx families the core trace logic is implemented to the Nexus standard. To undertake code-coverage analysis, a Nexus class 2 trace cell is adequate as all you need is detail of the program sequence on the individual core(s). If Branch History Messaging is used this can make the trace data very compact. Compared to standard trace data a reduction by a factor of 10 is realistic. Only PowerTrace II supports streaming from the Nexus trace port.

Streaming also works for all other processors/SoCs that are supported by TRACE32 and have a trace port.

**Code-Coverage for SMP-Systems**

TRACE32 also supports code-coverage analysis on SMP (symmetric multiprocessing) systems. For code-coverage it must be proven that an instruction was executed, which core was responsible for running the code is irrelevant. Figure 2 shows the results of code-coverage for two Cortex-A9 MPCores.

For statement and condition coverage, if only the fail-branch of a conditional statement was run the statement is highlighted in yellow and marked with “not exec”. The detailed coverage lists the specifics of how often each statement or each branch of the statement was run.

![Function coverage](image)

![Statement & condition coverage](image)

![Detailed coverage](image)

Fig. 3: Code-coverage analysis for an SMP system.