PLATFORM-BASED DESIGN
REPORT FROM THE FRONT

Sagheer Ahmad, Daniel Martin , Kambiz Khalilian,
Infineon Technologies 1730 N First St., San Jose CA, USA
Sagheer.ahmad@infineon.com
Daniel.martin@infineon.com
Kambiz.khalilian@infineon.com

Abstract. *The whole industry is now familiar with the concept of application-specific platform. An important class of topic is "the platform to build the platform". By that we mean the design methodology where embedded processors, memories, sub-systems, can be modified and reused to build a new application-specific platform (or SOC). This paper describes architectural considerations and unique design partitioning which allowed TriCore, a unified 32-bit processor, to be reused in more than 10 SOCs over a period of 3 years.*

1 Introduction

"Platform based design" refers to the design methodology that allows an OEM to develop a product (such as a DVD ROM player) in a minimum time by relying on a higher level of abstraction called "platform" [1,2].
A platform (or SOC) is made of software and hardware. The hardware part of a platform is itself composed of multiple cores, components and sub-systems. An important question comes to mind: "*what is the platform which helps to build the platform?"*
Since a high-performance embedded processor such as Infineon TriCore is the "essential core" of any platforms, it must make life easier for the platform integrator. For instance, it is major to provide flexibility in crucial design choices, such as memory and cache sizes, presence/absence of MMU and selection of sub-system parameters (bus type, frequency, etc.).
Each SOC has its own requirements in terms of performance, memory size, bus interface, and other features of the embedded processor cores. The ever-growing size and complexity of SOCs make it increasingly necessary to use high-performance embedded processors as hard macros.
By following certain architectural and design guidelines, we allowed TriCore to be re-used in more than 10 application-specific platforms over 3 years, most of them with minimal design effort.

2 Processor cores

There have been different approaches to achieve the reuse of processor cores [5,6]. Broadly, we can categorize such "configurable" core development into three categories.

2.1 RTL configurable

An RTL (Register Transfer Language) configurable core is developed with *generics* or *parameters* options in the RTL code [4]. It is synthesized based on the requirements of a particular SoC. This approach provides high degree of configurability, but requires a great amount of design effort, and is most suitable for low-performance cores.

2.2 Hard macro

In this "one core fits all" approach, the core is developed as hard macro (i.e. as a layout block). Such a core can only be used in a few SOCs that have the same core requirements. This approach provides almost no configurability, but can be reused without any extra efforts, and is most suitable for very high performance cores.

2.3 Modular

In the typical implementation, processor block is partitioned into sub-blocks on the basis of their functionality, e.g. CPU (central processing unit) is developed as one sub-block. In the modular approach, we partition the processor block on the basis of reusability, e.g., one sub-block MPU (micro-processor unit), which consists of CPU, is partitioned such that there are no snake (a combinatorial path between an output and an input) timing-paths in MPU. MPU is developed as a hard macro. Various configurabilty options are architecturally defined in the core special function registers. Other sub-blocks are either synthesized or developed as hard macro.
With this approach, high-performance cores can be reused with minimum design effort. We implemented TriCore using modular approach. A brief architectural introduction of TriCore is given in the following section.

2.4 TriCore

TriCore (Fig. 1) is a high-performance unified 32-bit RISC and DSP embedded processor with 16 and 32 bit instruction [2]. Its super-scalar architecture is optimized for real-time embedded applications and provides very fast interrupt response. The basic architecture is the load/store RISC architecture enhanced with Harvard type capabilities that are mostly found in DSPs. Among other things, its CPU provides [7]:

- A triple pipe, including zero-overhead loop pipe, super-scalar machine
- Hardware-maintained context switching
- 32-bit load/store Harvard architecture
- Shallow 4-stage pipeline
- Sustained single-cycle dual MAC

The MPU, which consists of the processor and code and data memory interfaces, is developed as a hard macro. In addition, code (program) and data memory, MMU, bus interface, debug interface, and interrupt control unit are defined at the architecture level such that any of these elements can be plugged or unplugged with the MPU.

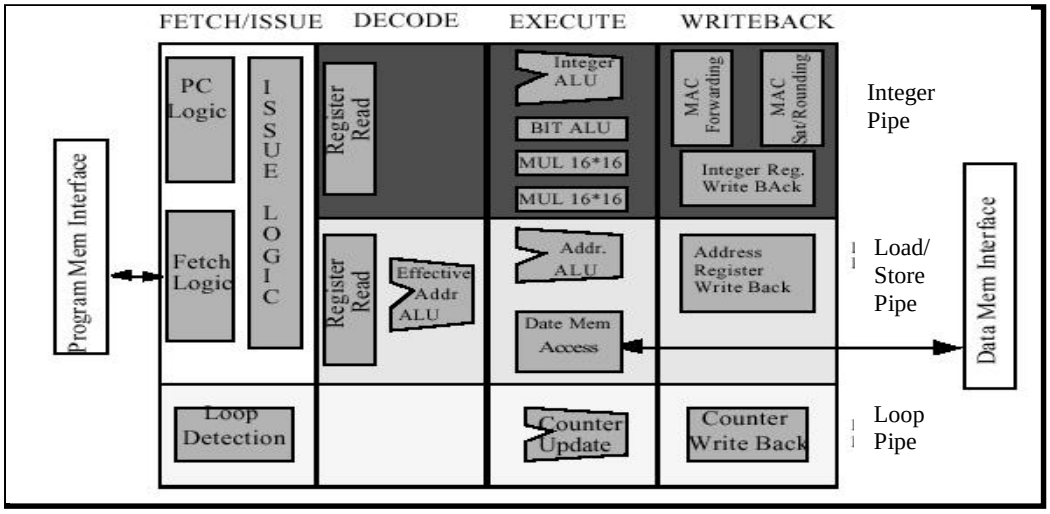

Figure 1: TriCore (Processor and memory-interfaces)

3 Design Partitioning

The CPU of a fast processor has normally custom-designed blocks in it. So, to re-use the core based on such a processor with minimum design efforts, we have to use the hard macro of the CPU. To enable the hard macro of a CPU to be re-used, it should be defined and designed such that the performance of sub-blocks of the core that interface with the outside (bus interface, interrupt control interface, etc.) do not require synthesis of the CPU. In other words, interface blocks should be independent of the CPU as much as possible, and we should be able to synthesize, place and route them based on the SoC requirements, independent of the CPU.

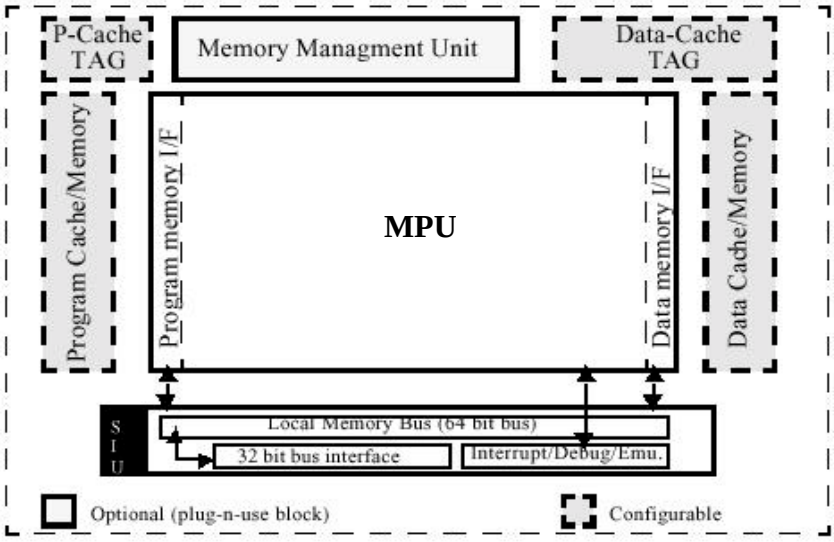

Figure 2: A general core block diagram

Traditionally, embedded processor design is partitioned into CPU, memory/cache interfaces, memories, and system sub-blocks (interrupt, debug etc). As shown in Figure 2,

we partitioned the processor design into MPU (microprocessor unit), SIU (system interface unit), and two separate memories etc. The idea is to have all common functionality, but no snake timing paths, in MPU.
MPU consists of the CPU, both code and data memory/cache interfaces, and BISTs for both code and data memories/caches. The MPU is developed as a hard macro, and the other blocks may be synthesized, or re-used as hard macro blocks if they meet the SoC placement requirements. If the core layout is not required to be rectangular, other blocks can be re-used, and we only need the code and data SRAMs as per the SoC specifications. For the other blocks, the interface design of the side abutting the MPU affects the maximum frequency CPU can achieve. In most cases, a few versions of the hard macros of other blocks can provide almost all the configurations, so we can develop them with the core. Then we can plug-in and use the modules to make a core according to the SoC requirements.

4 Design and Architectural considerations

The main considerations/blocks of the core that enable us to re-use it in a wide variety of SOCs are discussed below.

4.1 Bus Interfaces

We have two bus interfaces: a 64-bit local memory bus, and a 32-bit varying frequency system bus interface (fig. 3). A system designer can have memory or peripherals connected to one or both of the buses. Also, an external system bus can be interfaced to either the 64-bit local or 32-bit system bus.
To configure the MPU to use an external 64-bit or 32-bit bus, a core register can be programmed to route the external bus request to either the local or system bus. Such a register-controlled interface provides the SoC designer flexibility to have an external bus on either of the two buses without redesigning the core.
The system bus can be configured to run at the same or half of the MPU clock frequency, which is also controlled by a special function register. By having such a register-controlled bus interface, a hard macro of a core can accommodate a variety of bus interfaces.

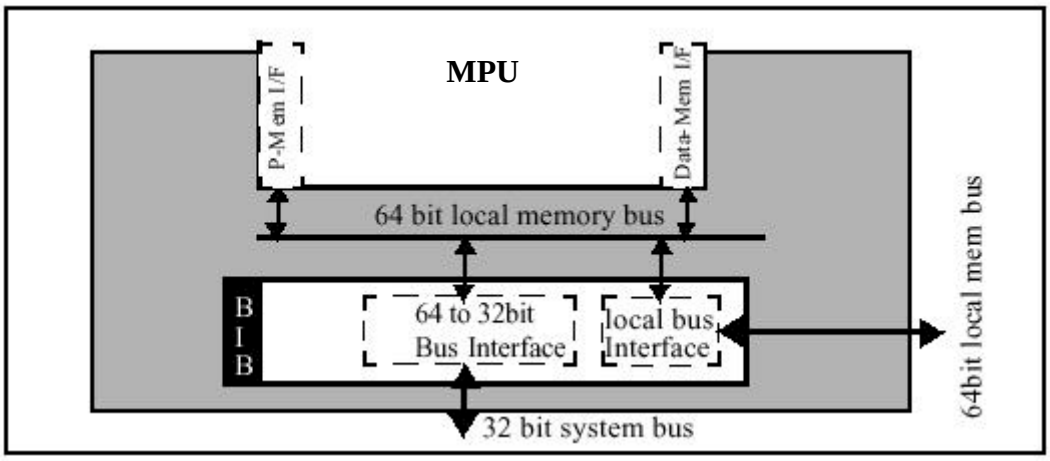

Figure 3: Flexible bus interfaces

4.2 Interrupt Control System

The multi-clock-cycle arbitration-sequence [7] based interrupt bus of the TriCore requires that all the peripherals requesting interrupt service sample the arbitration results at every arbitration cycle. To make the MPU frequency independent of the interrupt system frequency, the interrupt bus is designed to run at the same frequency as the MPU or slower. However, in a SoC, one can have a peripheral unit whose interrupt requesting node runs at the frequency of the peripheral. If we have several peripherals in the SoC, such an interrupt arbitration frequency may limit the frequency of the system as a whole. To avoid having the interrupt arbitration cycle limit the frequency of the peripherals, we have designed the arbitration cycle to be one or more interrupt system clock cycles, programmable through a special function register.

4.3 Memory Management Unit (MMU)

The MMU translates a virtual address into a physical address. The data load/store addresses are translated and fed into the Data Memory Interface, while the instruction fetch addresses are translated and fed into the Program Memory Interface. So to avoid having snake paths in MPU, the MMU should logically be part of the MPU hard macro. However, some SOCs require MMU and others don't. Having MMU, in MPU, significantly impacts the MPU area and performance. So MMU is designed as a separate hard macro, and can be plugged, unplugged, or disabled as follows:

- Plug-in the MMU: It impacts the speed and area. If not required, it can be disabled through an external pin.
- Unplug the MMU: Core still provides cache-ability for both code and data memories. It will improve the speed of the MMU and will reduce the area significantly.

4.4 Code and Data Memory Organization and Interface

One of the most important [4] requirements of core configurability is to be able to provide different size and type of cache and memory. In our approach, we use eight blocks of equal

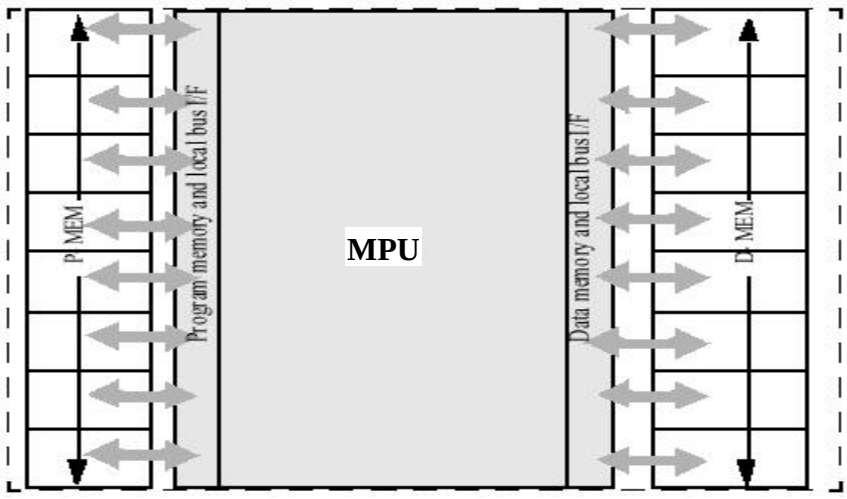

Figure 4: Memory organization and interface

size of synchronous SRAM for both, the code and data memories. The maximum memory size is limited by the minimum acceptable MPU frequency. In this core, we can provide, on each side (code and data), from 8K to 64K bytes in 8K increments, including caches. Out of this memory, cache (code or data) size could be configured up to 16K.
To be able to use the cache, we need to have appropriate TAG blocks (for different cache sizes, different TAGs are used). Part of the code or data memory is automatically used as cache memory if cache configuration is selected. Interface signals for each of the eight blocks are exactly the same. The concept of having eight similar blocks of memories is used because:
- We get a rectangular layout of MPU and memories together.
- Smaller memory blocks have faster access times, so having eight blocks enabled us achieve high speed.
- The height of the MPU layout is chosen such that we can accommodate eight memory blocks of typically available aspect ratios.
- Eight buffered clocks are provided at the interface inside MPU, so memory blocks can be abutted without further buffering of the clocks.
- For place-and-route, the memories are placed on each side of the MPU block, with precise pin placement to reduce insertion delay caused by wire delays. By using 0.18-micron technology, we have achieved 1.8ns of insertion delay with 0.2ns clock skew.

4.5 Clock Concept

We need three different clocks to be able to provide flexible-bus and interrupt interfaces.
- MPU clock: goes to all the MPU, memories, MMU, and TAGs
- Bus interface clock: goes to system interface unit (SIU)
- Interrupt system clock: goes to interrupt control system (part of SIU).

All three clocks are phase-synchronous and can be derived from the same PLL. During scan test, the PLL has to be switched to a special bypassing mode. This ensures that the frequency of all clocks is always identical.
The MPU has the clock-tree for MPU clocks going to the memories, MMU, and TAG. This enables to plug and use these blocks with the MPU. All clocks, including MPU, are supplied from the system interface unit. That provides the designer an option to rebalance the MPU clock with other clocks if he has to synthesize the SIU. The clock gating logic for MPU clock is designed in the MPU block.

4.6 Building A Specific Core

Each SoC requires a core with specific features. Examples of features are given below:
- Frequency.
- MMU or not
- Low-power silicon (high-VT), some high-speed silicon (low-VT)
- Redundancy logic for the memories
- Cache/memory ratio

In spite of all these differences, all our cores use the same functional building blocks. The main building block, the MPU, is the same and is re-used as a hard macro. Some other building blocks (e.g., MMU and TAGs) were also re-used as hard macros, and some (e.g., SIU) were available as soft macros. The TAG modules are designed, as hard macros, based on the size of the cache (4K, 8K, and 16K). Memory blocks are compiler generated for the specific process. If memories need to have redundancy, it could be added to each of the

eight blocks. Also, originally hard macros were developed in low-VT process (low threshold voltage). For SOCs based on high-VT processes, they were converted as explained now.

4.6.1 Converting from low-VT to high-VT without re-synthesis

This unique approach can be used to reduce the design effort needed by avoiding re-synthesis of the hard macros while targeting it to a different library (process), from low-VT to high-VT (or vice versa) of the same technology. Initially, we designed the MPU using Infineon's low threshold voltage library. Corresponding cells of both the libraries (low-VT and high-VT) have the same footprint. In fact, the same cells of the two libraries have the same layout, except for the poly layer. We can target hard macros designed based on one library to another without re-synthesizing. To achieve this, we designed our hard macros using the low-VT library by using only those cells that were common in both libraries. Although we used Apollo (Avanti's place-and-route tool), we believe most other place-and-route tools should be able to do this. From the Apollo database, we can dump-out the Verilog netlist with low-VT cells. In this Verilog netlist, we replaced the low-VT cells with high-VT cells and read this back into Apollo database. To avoid hold timing violations in the high- VT process, clock skew in the low-VT should be kept as low as possible.

4.6.2 Memories With Redundancy

A typical design of redundancy memory requires fuses (electrical or optical) to store the faulty address, fuse control, a mux (to select the register values in case of fault memory location) and its control, and a raw SRAM block.

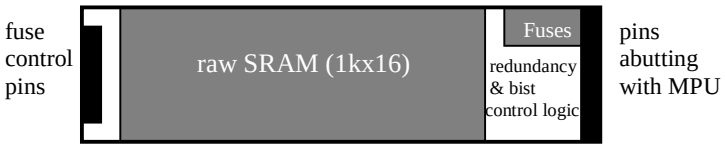

Figure 5: Layout frame with pin placement of SRAM with redundancy

To accommodate such a memory in our core, the memory block layout has to be such that the width of each memory block and the pin placement (interfacing with the MPU) are the same as required for a raw SRAM block. The extra pin introduced due to redundancy control should be placed on the opposite side (because, normally, memory layouts are completely blocking and we can not route on top of the memories).

5 Example of Application-specific Platforms

To illustrate the concept of hard macro configurable cores, we will describe three Application-Specific platforms (or SOCs) that re-used the core with different frequency, memory size, and other feature requirements.

5.1 Industrial Bus Controller

This highly integrated industrial controller [8], targeting general-purpose and industrial applications, was developed as a general-purpose industrial platform.

The TriCore-based core provided an external bus interface (usually a bottleneck of 32-bit devices) to be connected to the 64-bit bus interface to support the increasing demand for high-performance interfaces such as PCI and Ethernet.

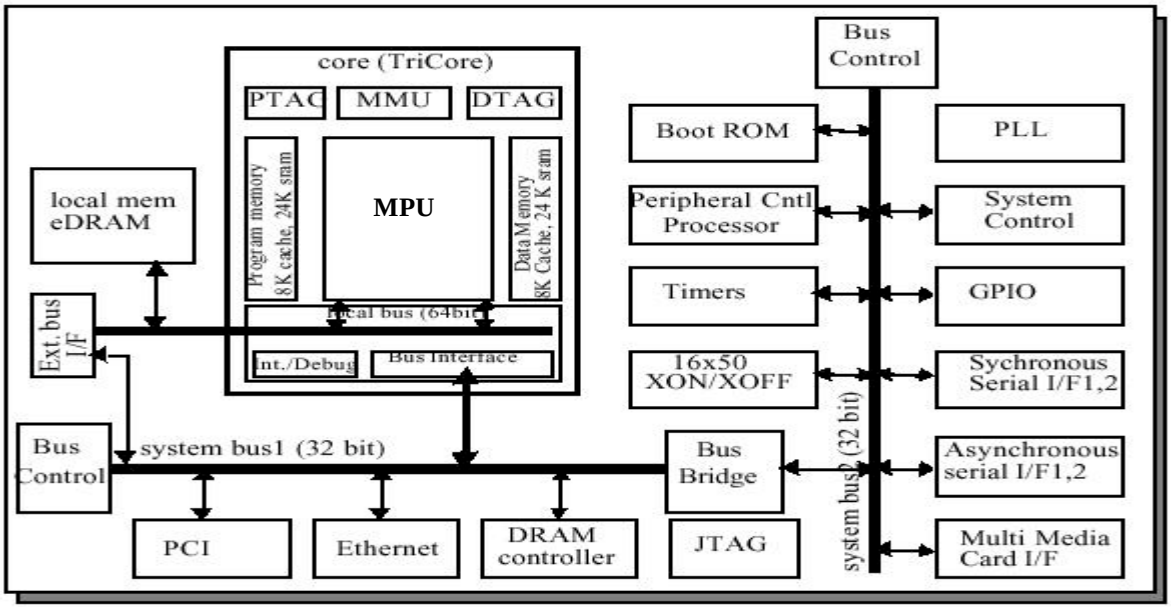

Figure 6: Industrial Bus Controller based on TriCore

Also this industrial platform required the MMU to be Windows CE-compliant. To develop the core for this SoC, we re-used hard macros of MPU, MMU, and 8K TAGs. The SIU was synthesized to provide the rectangular layout and optimized system bus. Eight memory blocks of 4 KB each were generated for both code and data memory.

5.2 Single-Chip Triple-Mode (UMTS/GSM/EDGE) Baseband IC

This cellular handset platform required a core with 16 KB of code and 64 KB of data memories with redundancy, including 8KB of cache on each side. The core also includes the MMU.

The chip has software stored in a large local memory (eDRAM). So this eDRAM was connected to the CPU via a fast and wide bus, to provide sustained code bandwidth. The chip also had a memory connected to 32-bit bus, which runs at half the frequency of the CPU. So the core provided an interface with this memory. Since the chip design was based on high-VT, so we converted our hard macros into high-VT. Due to the many peripherals and the large size of the chip, delays of the interrupt arbitration bus signals was large, so the interrupt bus arbitration cycle (interrupt system frequency is at half of MPU frequency) was required to be two interrupt clock cycles.

5.3 Multibeam DVD Read/Write IC

This DVD platform required the core to be without MMU but with 32 KB of code and 32 KB of data memories, including 8 KB of code cache. Since MPU has been provided with pins that can be tied to the appropriate values to configure the cache size, the cache size was selected as needed.

Also, we used hard macro of 8K TAG; however, it resulted in a non-rectangular core. The resulting "whitespace" in the layout is not blocked and is used by the SoC. Since the width of the core layout is determined by memory sizes.
SIU was synthesized to match with the width of core layout. Because the memory size and type of the DVD platform and the industrial bus controller were the same, we could re-use the SIU also. In summary, to develop the core for this SoC, we had to integrate the MPU, PTAG, SIU, and memories.

6 Conclusion

Increasing numbers of SOCs are using high-performance embedded processors. Each SoC has its own requirements of performance, memory size, bus interface, and other core features. By following certain architectural and design guidelines, processors can be developed as configurable, which can be re-used with minimum design effort. In this modular core approach, the CPU and other common functions of the core are developed as a hard macro - MPU. The pins at the MPU boundary and core special function registers are provided, wherever possible without compromising performance, to control the various configurability options in the hard macro. MPU hard macro is developed with dedicated memory interfaces for eight memory blocks on the code and data sides. The MMU is developed as a hard macro, and is integrated into the core if required. The pins, provided at the MPU hard macro, control the size of the cache. TAGs are designed based on the size of the cache (4K, 8K, and 16K). A system interface unit (SIU), which consists of bus interfaces, interrupt interface, debug and trace units, is synthesized. The SIU, having three clock domains, is optimized for required bus and interrupt system interfaces.

References

[1] Grant Martin & all : Surviving the SOC revolution - A guide to platform-based design, Kluwer Academic Publishers 1999.

[2] Keating M. and Bricaud P.: Reuse Methodology Manual for SoC designs, Kluwer Academic Publishers, 1998.

[3] Ober R.: A fast unified processor core, embedded processor forum 2001, San Jose, USA.

[4] Gogolewski M, Miller W.: Configurable Memory Processor Core, DesignCon 2002, Santa Clara, USA.

[5] Rabaey J.M.: A system level perspective on Reuse, IP98 - March 24 1998.

[6] Runner J.S., Sanaka V., Yu E.: Building an infrastructure for IP reuse, EE Times, May 15 2000.

[7] TriCore - Architecture Manual v1.3.1, April 2001, published by Infineon Technologies.

[8] TC11IB - Highly Integrated Microcontroller, Product Brief, Infineon Technologies, 2000.