Chapter 9
CRISP: Cutting Edge Reconfigurable ICs for Stream Processing

Tapani Ahonen, Timon D. ter Braak, Stephen T. Burgess, Richard Geißler, Paul M. Heysters, Heikki Hurskainen, Hans G. Kerkhoff, André B.J. Kokkeler, Jari Nurmi, Jussi Raasakka, Gerard K. Rauwerda, Gerard J.M. Smit, Kim Sunesen, Henk van Zonneveld, Bart Vermeulen, and Xiao Zhang

Abstract The Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project aims to create a highly scalable and dependable reconfigurable system concept for a wide range of tomorrow's streaming DSP applications. Within CRISP, a network-on-chip based many-core stream processor with dependability infrastructure and run-time resource management is devised, implemented, and manufactured to demonstrate a coarse-grained core-level reconfigurable system with scalable computing power, flexibility, and dependability. This chapter introduces CRISP, presents the concepts, and outlines the preliminary results of a running project.

9.1 Project Partners

1. Recore Systems, The Netherlands (coordinator)
2. University of Twente, The Netherlands
3. Atmel Automotive GmbH, Germany
4. Thales Netherlands, The Netherlands
5. Tampere University of Technology, Finland
6. NXP Semiconductors, The Netherlands

- Project Coordinator: Paul M. Heysters, Recore Systems, The Netherlands
- Start Date: 2008-01-01
- End Date: 2010-12-31
- EU Program: 7th Framework Programme, FP7 ICT-2007.3.4, STREP Project No. 215881
- Global Budget: 4.4 M €

K. Sunesen (✉)
Recore Systems, Enschede, The Netherlands
e-mail: Kim.Sunesen@recoresystems.com

- Global Funding by EU: 2.8 M €
- Contact Author: Kim Sunesen (Recore Systems), Email: Kim.Sunesen@recoresystems.com

9.2 Introduction

Streaming applications have high market potentials and drive demands for reconfigurable platform chips [1, 2]. It is becoming increasingly more difficult to predict which applications are going to be successful in the future. For this reason, application providers are increasingly interested in programmable platform chips, which enable to anticipate on expected market trends and offer flexibility, if the market develops differently [3].

The Cutting edge Reconfigurable ICs for Stream Processing (CRISP) project develops a scalable and dependable reconfigurable multi-core system concept that can be used for a wide range of streaming applications in the consumer, automotive, medical and defense markets. The envisioned platform solution includes a massive multi-core processing architecture in combination with innovative design-time and run-time tools. CRISP addresses optimal utilization, efficient programming and dependability of reconfigurable many-cores for streaming applications. The main objective is to create a General Stream Processor (GSP) for tomorrow's streaming applications. There are many challenges in developing massive multi-core platform solutions. CRISP addresses the fundamental underlying problems. The concerns driving this project are indirectly attributable to the miniaturizations of semiconductor technology and translate into three essential questions:

How can the intrinsic processing potential of a massive multi-core architecture be exploited optimally for a wide range of streaming applications?

How can multi-core systems be programmed efficiently?

How can large multi-core integrated circuits using deep submicron semiconductor processes be made reliable and self-repairing?

The CRISP project uses a holistic and pragmatic applied research approach to find answers to these questions. The project is organized around four central themes:

Streaming applications – Targeted applications range from low-end consumer electronics and automotive applications to demanding high-end medical and defense applications.

General Stream Processor (GSP) – The GSP is a dynamically reconfigurable many-core platform for streaming applications. The scalable architecture targets flexibility, high performance, low power and a small footprint.

Run-time mapping – Computational resources in a many-core can be efficiently utilized by resource allocation at run-time. This also enables upgrading, bug fixing and hardware fault diagnosis and repair at run-time.

Dependability – Dependability and yield of deep-submicron chips are improved using new techniques for static and dynamic detection and localization of faults and (dynamically) circumventing faulty hardware.

Results of CRISP will include concrete hardware manufacturing as well as software development of tools, systems, and applications. This chapter dedicates a section to each of these themes. While the CRISP project is still running at the time of writing, preliminary results are achieved for all themes. The CRISP approach demonstrates the synergy of the four themes.

9.2.1 Related FP Projects

The CRISP project can be seen as a successor of the 4S ("Smart chipS for Smart Surroundings") FP6 project [4–6]. In 4S, reconfigurable computing proved to deliver high performance while being energy efficient, flexible, programmable, and run-time adaptable. More details can be found in the 4S project chapter of this book [4]. The results of 4S also revealed new research topics concerning scalability of multi-core systems and dependability of deep submicron technologies. On this background, three 4S project partners Atmel, Recore Systems, and University of Twente joined with NXP Semiconductors, Thales Netherlands, and Tampere University of Technology to form the CRISP Consortium to break new grounds in scalable and dependable high-performance computing using dynamically reconfigurable many-core platforms.

9.2.2 Chapter Structure

The remainder of this chapter is organized along the themes of CRISP. The next four sections introduce and explain the concepts of streaming applications including the specific CRISP demonstrators, the CRISP General Stream Processor, run-time resource management, and dependability. Before concluding, we present preliminary results of CRISP and take a look towards the future.

9.3 Streaming Applications

CRISP research focuses on efficient embedded hardware and software solutions for stream processing used in a wide range of streaming applications including baseband processing for wireless communications, multimedia, sensor, medical image, and intelligent antenna signal processing. Streaming applications can be modeled as data flow graphs with streams of data items (the edges) flowing between computation kernels (the nodes) [1, 7]. Streaming applications inherently hold locality and concurrency properties. Key characteristics include:

- High data-throughput between computing intensive kernels;
- Local processing on data items;
- Hard real-time throughput guarantees;
- Semi-static operation life-time.

The kernels and communication topology of the data flow graph, the data bandwidth between kernels, data item sizes and real-time guarantees vary with every application.

In the consumer market there are strong trends towards integrating more and more streaming applications into a single device, ever faster changing standards and algorithms demanding ever more processing power, as well as a trend of rising upfront investments for silicon manufacturing. In the medical, defense, and security markets, there is a trend towards reducing development and maintenance costs as well as using state-of-the-art technology by choosing commercial off-the-shelf components. Together these trends fuel the need for reconfigurable ICs that combine some of the flexibility known from Field Programmable Gate Arrays (FPGAs) with the speed, size, and energy advantages known from Application Specific Integrated Circuits (ASICs).

In CRISP, concrete streaming applications are worked out for digital beamforming and Global Navigation Satellite System (GNSS) reception. Digital beamforming is chosen as a typical example of a high-end application with high throughput and computing demands. Digital beamforming is used in defense and space applications but is also becoming increasingly popular in consumer electronics applications such as femto-cell and wireless access gateways. Satellite navigation systems have become omnipresent in e.g. cars, smart phones, and wrist watches. GNSS is chosen as a typical example of a low-end consumer application. The ambition of the CRISP platform is to scale from low-end to high-end applications using the same reconfigurable System-on-Chip (SoC) template. Both applications play their role in validating the success of the CRISP approach.

9.3.1 Digital Beamforming

Digital beamforming is used in an increasing range of products like sonar systems, radar systems, radio astronomy telescope systems, and base stations for wireless telecommunications. In the CRISP project, the digital beamforming application is derived from the radar field [8, 9] where requirements are demanding in terms of both data throughput and processing power. For instance, a system with 64 antenna receive channels typically has input rates of tens of Giga bytes per second and requires several Giga Multiply Accumulate (MAC) operations per channel.

Figure 9.1 depicts the functional architecture of a beamformer system. In the analog part, signals are received from multiple antennas and converted into digital signals. In the digital part, antenna processing is applied to each signal for calibration

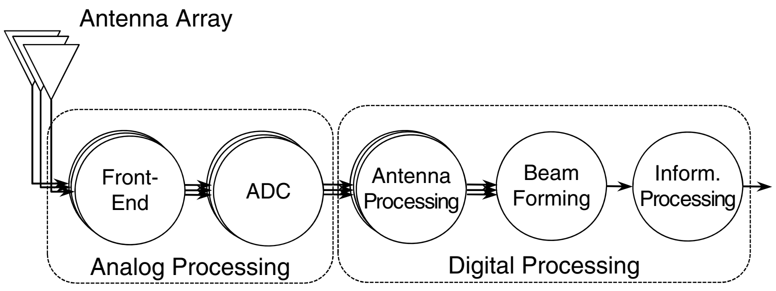


Fig. 9.1 Functional architecture of a digital beamformer system

or equalization and beamforming combines signals into beams that are further processed. Below, in the section on preliminary results further details on how the digital processing of the beamformer system is mapped on the CRISP platform.

9.3.2 Global Navigation Satellite System Application

Satellite navigation applications of today range from cheap, single-frequency receivers embedded in mobile phones to expensive, multi-frequency, centimeter-accurate scientific receivers. In the CRISP project, the GNSS application is specified and designed to support the existing U.S. based NAVSTAR Global Positioning System (GPS) [10] and the future Galileo (European satellite navigation system) [11] signals transmitted in the L1 frequency band.

Three main blocks in any GNSS receiver can be identified as (i) a radio front-end for analogue signal processing (shown as Radio in Fig. 9.2), (ii) a digital baseband processing part for navigation data decoding and signal time-of-arrival measurements (shown as Digital Baseband Processing in Fig. 9.2), and (iii) navigation calculus to determine position, velocity and time (PVT) (shown as Navigation Task in Fig. 9.2) [12].

The digital baseband processing tasks are: acquisition (search of satellites) by using long FFTs and tracking, data decoding and signal time-of-arrival measurements by means of serial correlation. These receiver domains are present in all GNSS receivers through the whole range from inexpensive mass market devices to high-end scientific receivers. In the CRISP implementation, four to eight acquisition or tracking processes are expected to be active at any point in time.

Most of the current state-of-the-art GNSS receivers are using dedicated hardware for digital signal processing (ASIC technology), but the trend is going towards Software Defined Radio [13]. The Software Defined Radio approach is motivated by the challenges faced due to the modernization efforts in satellite navigation systems. The European Galileo system together with systems from Russia (GLONASS) and China (Compass) are emerging to compete with U.S. based GPS, which is also undergoing a modernization program. The specifications of new systems are still evolving

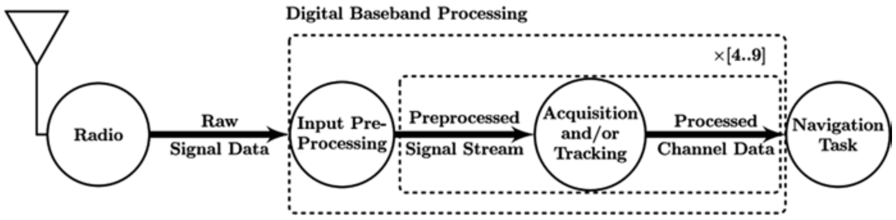


Fig. 9.2 Illustration of GNSS application tasks

and also the algorithm development in the field of multi-system reception is active. Thus, future receivers should have high computational power and flexibility for updates. A scalable platform with multiple reconfigurable processing tiles meets both of these requirements.

9.4 General Stream Processor (GSP)

The General Stream Processor (GSP) is a scalable platform template for performing virtually any streaming application, from low-end consumer applications to high-end applications. The GSP architecture is essentially a heterogeneous many-core system-on-chip (SoC) containing general purpose processor cores (e.g. ARM9™), reconfigurable cores (e.g. Xentium®), and memory tiles.

The proposed GSP architecture is designed with the principles of locality-of-reference and concurrency in mind. Hence, different levels of storage are defined in the CRISP many-core architecture; local data memories are incorporated in Xentium processing tiles and distributed on-chip memory is available in the GSP by means of the Smart Memory Tiles. Moreover, concurrent processing of computation kernels in the streaming applications can be performed on the many parallel Xentium processing tiles in the GSP. The Network-on-Chip and the distributed on-chip memories provide the communication infrastructure in the GSP to manage the data streams between the computation kernels in the streaming applications.

Within the scope of the CRISP project, a scalable GSP platform is built and subsequently demonstrated using many multi-core devices that are connected to each other to create a large many-core system-of-chips.

The GSP demonstrator is built from two kinds of chips: a *General Purpose processor Device* (GPD) and *Reconfigurable Fabric Devices* (RFD). The GSP depicted in Fig. 9.3 combines one GPD and five RFDs to build a many-core with 46 processing cores: one ARM® processor and 45 Xentium DSP cores. The Xentium DSP core is explained in more detail below. The inter-chip connection is handled by a dedicated Multi-Channel Port (MCP) chip-to-chip interface that extends the Network-on-Chip (NoC) of the RFDs across multiple chips. The MCP enables off-chip communication and is instrumental in demonstrating scalability as it allows NoC communication to extend seamlessly across chip boundaries.

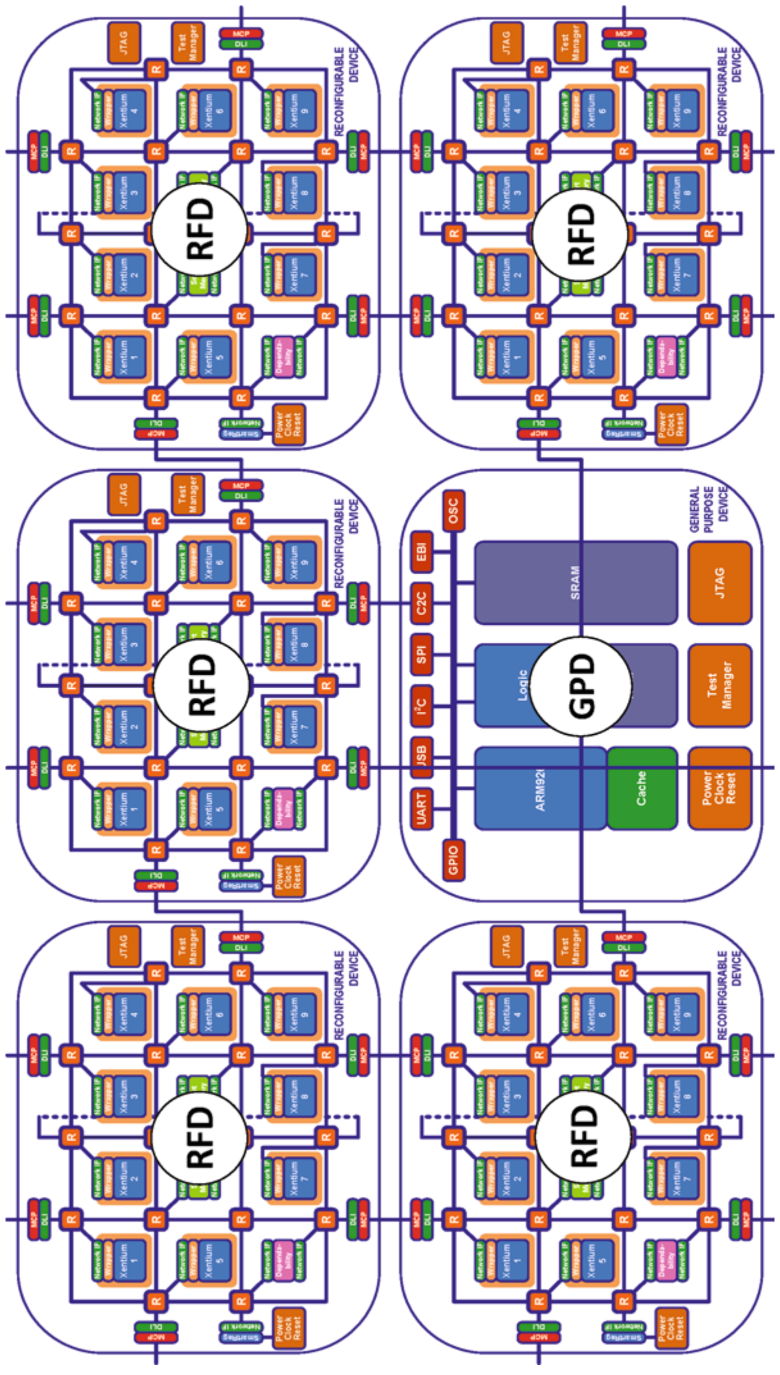


Fig. 9.3 Illustration of a GSP demonstrator connecting one GPD chip and five RFD chips in an off-chip network. This GSP connects forty-five Xentium DSP cores and one ARM® core in one network. Fig. 9.4 and Fig. 9.5 depict, respectively, the GPD and the RFD in more detail

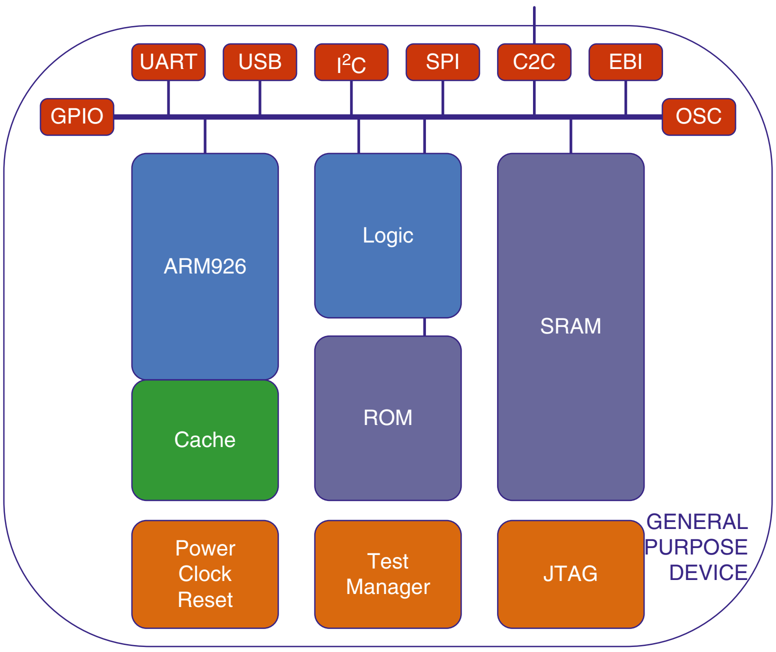


Fig. 9.4 General purpose processor device

9.4.1 General Purpose Processor Device (GPD)

The GPD is based on a traditional bus-based architecture and is depicted in Fig. 9.4. It is designed to run at a clock frequency of 200 MHz, and contains an ARM926 processor, ROM, SRAM and a wide range of peripherals. Among the common interfaces of the GPD, special focus has been given to the chip-to-chip (C2C) interface. In the GSP demonstrator (refer to Fig. 9.3) the C2C interface bridges the GPD with the packet-switched NoC of the RFDs. The GPD has full access to all resources of these RFDs using the C2C interface.

9.4.2 Reconfigurable Fabric Device (RFD)

The RFD contains the reconfigurable hardware blocks of the GSP platform. The RFD is a tiled processor comprising a grid of tiles interconnected via a packet-switched NoC. It is a complete SoC with I/O, testing infrastructure, and clock and reset management. In CRISP, a twelve-tile SoC architecture is implemented. The diagram of the twelve-instance RFD is given in Fig. 9.5. The RFD contains the following operational components that can be used to implement streaming DSP applications:

- 9 Xentium processing tiles
- 2 Smart Memory Tiles (SMT)

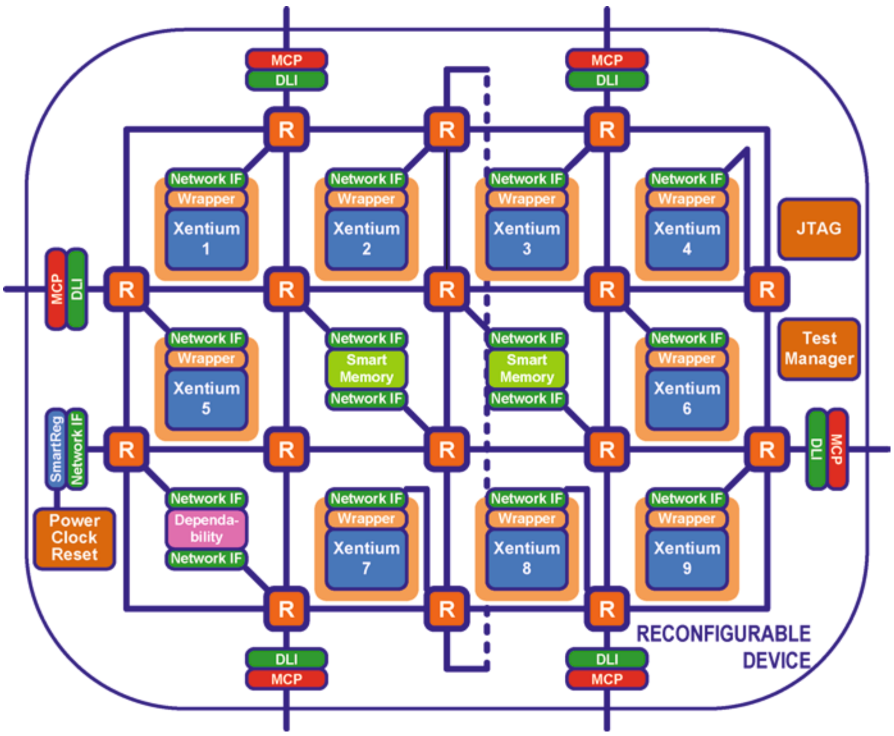


Fig. 9.5 Reconfigurable fabric device

- Dependability Manager tile (DM) + Infrastructural IP
- 6 Multi-Channel Ports (MCP) + Die Link Interfaces (DLI)
- Packet-Switched Network-on-Chip

The main communication infrastructure in the RFD is provided by the NoC. All Xentium processing tiles and memory resources are accessible via the NoC. Moreover, the NoC can be used to configure the clock manager of the RFD, and to access scan chains and memory Built-In Self-Test units of the Xentium tiles.

9.4.2.1 Scalable On-Chip and Off-Chip Communication Infrastructure

A packet-switched NoC connects the tiles in the RFD. The NoC provides means for on-chip scalability of the GSP template. Hence, larger many-core SoCs can be designed by increasing the number of routers in the NoC.

In the GSP demonstrator, off-chip scalability is addressed by extending the NoC across the RFD chip boundaries using dedicated MCP interfaces. The MCP provides a transparent off-chip interface to bridge two RFDs or to connect the RFD with the C2C interface of the GPD. The MCP interfaces enable prototyping of large many-core systems-of-chips as depicted in Fig. 9.3.

Scaling the packet-switched NoC to larger dimensions results generally in increased NoC routing overhead because of the increasing number of hops in the NoC. Therefore, the CRISP project researches hierarchical addressing and NoC routing. The DLI provides means for hierarchical addressing to the GSP.

The NoC is designed to run at a clock frequency of 200 MHz (in 90 nm CMOS) and provides an on-chip bandwidth of 6.4 Gbps in each direction of each link. The data rate of the external MCP interface is 100 Mbps per external pin using a double data rate transfer scheme to reduce signal distortion at PCB level. With an input and output data width of 8 bits, the bandwidth of the MCP interface is 800 Mbps in each direction. The difference in bandwidth with the external inter-chip links results in higher communication costs. Hence, on-chip communication between tiles is preferred over off-chip communication between chips. The run-time resource management software takes those increased communication costs into account in the resource allocation.

9.4.2.2 Xentium Tile

The Xentium tile is a programmable high-performance fixed-point digital signal processing (DSP) core. High-performance and energy-efficiency are achieved by optimizing parallel operation at instruction level.

All communication with the Xentium tile is performed using memory-mapped I/O; all modules in the Xentium tile are given a dedicated address range in the Xentium memory map.

The Xentium tile is designed to operate at a clock frequency of at least 200 MHz (in 90 nm CMOS technology, worst case conditions). If required, methodologies from simple clock gearing up to sophisticated dynamic voltage and frequency scaling (DVFS) can be implemented locally in the Xentium tile to reduce the power consumption of the processing tile. The area requirement of the tile equipped with 16 kBytes of data memory and 8 kBytes of instruction cache is ~1.8 mm^2 in 90 nm CMOS technology.

The core modules of the Xentium tile are the Xentium core, tightly coupled data memory and a NoC interface as shown in the block diagram in Fig. 9.6. Moreover, the Xentium tile contains additional logic that is used to control scan chains and memory Built-In Self-Test at run-time. The additional logic is part of the Dependability Infrastructural IP, described below in the section on Xentium tile dependability, where it is referred to as Xentium Tile Wrapper.

Xentium datapath: The Xentium datapath contains parallel execution units and register files. The different execution units can all perform 32-bit scalar and vector operations. For vector operations the operands are interpreted as 2-element vectors. The elements of these vectors are the low and high half-word (16-bit) parts of a 32-bit word. In addition several units can perform 40-bit scalar operations for improved accuracy. All operations can be executed conditionally. The Xentium datapath provides powerful processing performance:

- 800 16-bit Mega MACs per second or
- 400 32-bit Mega MACs per second or
- 400 16-bit complex Mega MACs per second

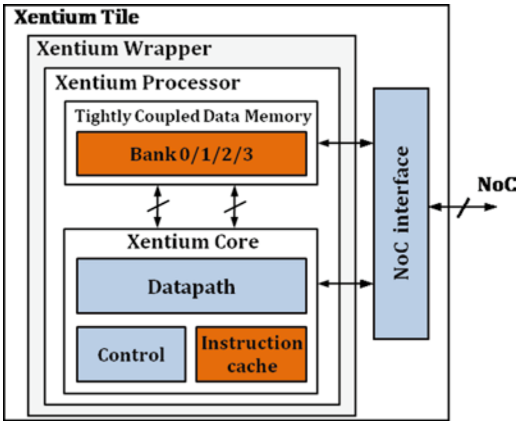


Fig. 9.6 Xentium tile

Xentium control: The control block in the Xentium core performs instruction fetching and decoding, and controls the execution units in the datapath. Instructions are fetched from Xentium-external memory (e.g. Smart Memory Tile) and are stored in the Xentium instruction cache. The programmer can indicate that a section of a Xentium program has to be prefetched by the control to ensure that the instructions of that section of the program are cached. This prevents cache misses during execution, which makes the execution time of the prefetched section of the program predictable (provided that the execution time of the data loads and stores in the section are predictable). The prefetch mechanism provides means to reconfigure the Xentium tiles in the RFD with the required DSP functions. The run-time resource management software (running on the GPD) can, for instance, issue a prefetch command in order to force one of the Xentium tiles in the RFD to reconfigure with new DSP functionality.

Xentium tightly coupled data memory: The tightly coupled data memory is organized in parallel memory banks to allow simultaneous access by different resources. The data memory can be simultaneously accessed by the Xentium core as well as by the Xentium NoC interface. By default the data memory in the Xentium tile is organized in four banks of 4 kbytes each, implemented using SRAM cells. The size of the memory banks is parameterizable at design-time.

9.4.2.3 Smart Memory Tile

In the RFD, the Smart Memory Tiles (SMT) provide shared memory that is accessible through the NoC. The SMT contains parallel memory banks with a total memory size of 64 kBytes per memory tile in the RFD.

Besides being accessible with random access, the SMT has multiple reconfigurable Address Generation Units (AGU). The AGU can generate a sequence of addresses autonomously. Multiple AGUs can be linked, giving the possibility to configure FIFO functionality or implement elastic buffering.

Memory tiles are key to efficiently implement reconfigurable many-core SoCs:

- multiple memory tiles are used as distributed data memory to reduce the required bandwidth to a single shared memory;
- memory tiles are used as local code caches to store the binaries of reconfigurable tiles in the SoC;
- memory tiles can function as elastic buffers in the SoC (e.g. to buffer input or output data streams from I/O interfaces).

9.5 Run-Time Resource Management

Due to the scalability of the GSP platform, some configuration effort is required to load an application onto the processing elements. This includes the assignment of tasks to specific Xentium cores, and configuring the routing tables of the NoC with routing information. When a programmer performs these steps (manually) at design-time, an assumption must be made on the availability of resources. This results in potential conflicts, when resources are in use by other applications, or when resources are unavailable due to hardware faults. A more extensive design-time analysis can only consider a limited number of use-cases, and event- or user-triggered actions are difficult to anticipate on. Any change in the application or in the platform that is not captured by the design-time analysis, results in incorrect or unpredictable behavior.

Even in application specific platforms, where less flexibility is required, resource allocation at run-time may have additional advantages in fault tolerance. For example, when faults are detected by a postproduction test, otherwise disabled spare parts may be used to replace the malfunctioning components; this is called static redundancy [14]. A chip then may still be usable if the number and type of faults are non-critical. This approach is used in the manufacturing process of e.g. the Cell processor, where not all eight Synergistic Processing Elements are required, resulting in a higher production yield [15]. Such scenarios only work when the application mapping to the architecture can deal with these deficiencies. On the other hand, safety critical systems often use online fault detection mechanisms. With redundant resources available, the system may continue its operation if a detected fault can be isolated. Thus, to provide support for fault tolerance, a run-time resource manager should be in place to account for the free, allocated and faulty resources in the system.

Larger systems, such as heterogeneous computing clusters, use a distributed memory model to overcome the high communication overhead of shared memory systems. Such systems often use middleware to present the system in a homogeneous manner. However, distributed memory multi-processor systems are in general heterogeneous [16], and it may be quite expensive in terms of performance and energy consumption to expose them as homogeneous systems. Keeping the property of homogeneity may even be considered impossible, due to asymmetric resource allocation for multiple applications simultaneously running in the system. Considering parallel programming, the most complex platform is a multi-user, heterogeneous cluster made up of processors of different architectures, interconnected via a heterogeneous communication network [16].

The GSP platform incorporates a distributed memory model, and besides the efficiency arguments, the Xentiums are not designed to run any middleware. Therefore, the GPD manages the resources of the connected RFDs in a centralized manner. Each application requests the amount and type of resources required for its execution. The resource manager determines whether these requests can be fulfilled. The resource manager must shield the tasks that are already present on the platform from the interference caused by newly started applications. A resource management policy [17] should enforce such conditions:

- *admission control* – an application is only allowed to start if the system can allocate, upon request, the resources the application required to meet its performance constraints;
- *guaranteed resource provisions* – the access of a running task to its allocated resources cannot be denied by any other task.

If an application cannot be added to the system without a violation of this policy, then the resources for the application will not be allocated. Hence, in that case the application is refrained from execution on the system by the run-time resource manager.

9.5.1 The Resource Allocation Problem

The resource allocation problem concerns two dimensions: the spatial and temporal dimension. Explicitly considering the (relative) location of resources within a platform may avoid inefficient resource allocations, where efficiency may be measured in the amount of resources being allocated, or in energy consumption of the platform. Orthogonal to the spatial dimension, we consider the temporal dimension. This does impose some uncertainty, as we assume that we do not know in advance which applications are started or stopped at a certain time. Therefore, optimal solutions cannot be guaranteed because it depends on future events. The resource management algorithms thus resort to finding feasible solutions, while optimizing towards secondary objectives, such as minimal energy consumption or highest performance. Additionally, in the case that multiple applications may be executed simultaneously on the platform, a reasonable platform state has to be maintained that allows for more than one resource request to be fulfilled. Given these properties and the complexity of the problem, heuristics are used to tackle the problem.

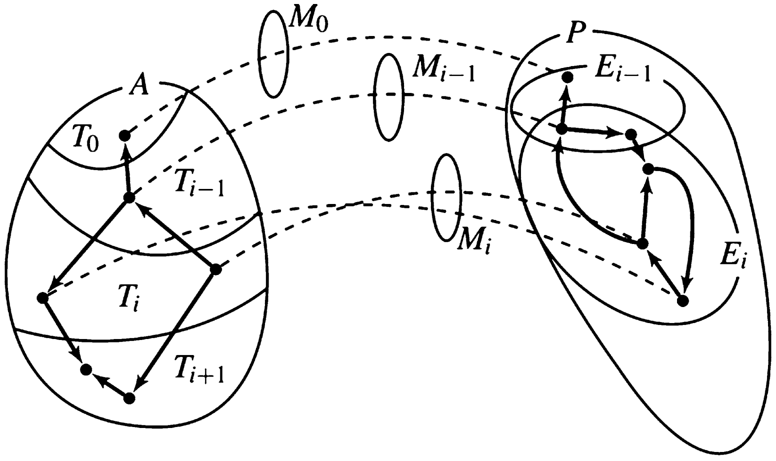


Fig. 9.7 The resource allocation problem is partitioned in multiple sub-problems, using the structure of the task graph

A graph representation of applications and platform is used to reason about the connectivity of the resource demands and provisions, respectively. In spatial resource allocation, the problem is to find specific locations to fulfill the resource requirements of the tasks T and communication channels C in an application $A=(T,C)$. A platform $P=(E,L)$ provides resources through the processing elements E, which are connected with the links $L \subseteq E \times E$. In Fig. 9.7, the graph of tasks T_i describes an application A, which needs to be mapped on a set of resources E_i composing platform P. Mapping M denotes the specific assignment of tasks to resources. Each task needs to communicate with at least one other task in the same application; otherwise, that task would compose a problem instance of its own. For each pair of communicating tasks, the communication infrastructure has to provide a communication route with enough bandwidth between their assigned processing elements. The combination of these problems may be formulated as a constraint optimization problem.

The topological aspect limits the number of possible task assignments. Therefore, exploiting the topology of the application and the topology of the platform vastly reduces the search space. The resource manager incorporates a heuristic that uses divide-and-conquer to break the resource allocation problem into sub-problems of variable size, depending on the density of the task graph.

The sub-problems that have to be solved are instances of the common generalized assignment problem, which is NP-hard [18]. Taking the communication requirements between tasks into account, for each subset of tasks T_i a subset of candidate elements E_i is selected. For every $e \in E_i$, the cost of mapping each task $t \in T_i$ is calculated. This procedure is applied to each subset of tasks T_i in application A. Thus, following the topology within application A, mapping M is incremented by each mapping M_i of tasks T_i to a set of resources E_i. In-depth details about the implementation are explained in [19]. If all tasks in the application can be mapped to a certain location in the platform and if sufficient communication resources are available, the application may be started.

One important aspect of resource allocation is to maintain clear and consistent state information of the GSP. Therefore, the resource management system software needs to know which cores and links in the SoC are available to map tasks. By using feedback from dependability software as discussed in the next sections, the resource manager receives feedback regarding broken links and erroneous cores in the GSP [20].

9.6 Dependability

The dependability of large scale multi-processor systems-on-chip is becoming an important concern, especially when the SoCs are used in mission-critical applications. Dependability is the extent to which a system can be relied upon to perform its intended functions under defined operational and environmental conditions at a given instant of time or given interval.

In the next sections, the CRISP approach for improving the dependability of the reconfigurable multi-core GSP is outlined. Dependability in the GSP focuses mainly on two aspects: making the NoC more dependable and improving the dependability of Xentium tiles. The first aspect ensures the correct operation of the NoC, which is a property that the second aspect heavily relies upon.

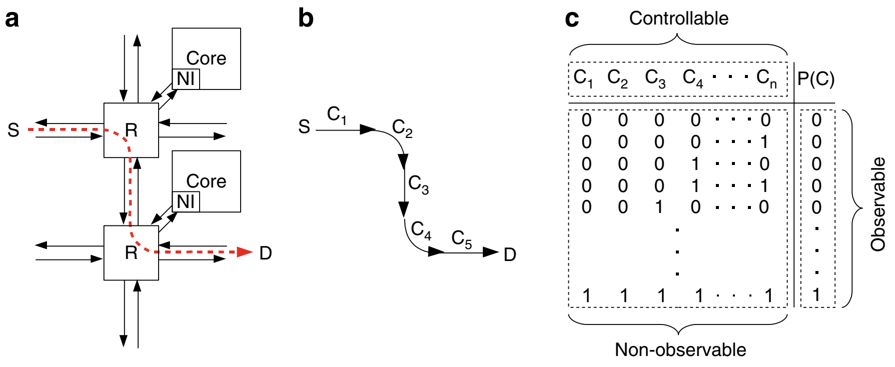
9.6.1 Network-on-Chip Dependability

The NoC dependability test is a GPD-hosted utility for diagnosing (i.e. detecting and locating) faults within the communication infrastructure of the GSP [21]. By using a software-only approach and solely using the existing NoC for test access, the need for specialized Design for Test structures within the NoC is eliminated. This saves silicon area and increases the ease with which the system may be deployed.

9.6.1.1 Network Fault Modeling

A model has been adopted which describes the NoC infrastructure in terms of inter-die and inter-router connections (link components) plus the switch components inside the router (intra-router routing paths), any of which may be generally referred to as a (path) component C. Accordingly, an arbitrary path through the network can be represented by a series of interconnected path components, C_1 to C_n.

For example, the network path illustrated in Fig. 9.8a is modeled by the five components C_1 to C_5 labeling the graph in Fig. 9.8b. Observe that C_1, C_3, and C_5 are link components whereas C_2 and C_4 denote switch components. The overall fault status of



C_1	C_2	C_3	$C_4 \cdots C_n$		$P(C)$
0	0	0	0 · · ·	0	0
0	0	0	0 · · ·	1	0
0	0	0	1 · · ·	0	0
0	0	0	1 · · ·	1	0
0	0	1	0 · · ·	0	0
			·		·
			·		·
			·		·
1	1	1	1 · · ·	1	1

Fig. 9.8 Modeling of the NoC **(a)** Arbitrary path through the network **(b)** Path represented in terms of a connected component graph **(c)** Controllability and observability of path parameters

a path, *P(C)*, is expressed as the logical AND of each constituent path component as given in Eq. 9.1, where *C* takes on the value of '0' for faulty or '1' for good:

$$P(C) = C_1 \bullet C_2 \bullet C_3 \bullet C_4 \bullet .... \bullet C_n \tag{9.1}$$

Therefore, $P(C)=1$ denotes a fault-free path whereas $P(C)=0$ indicates that at least one path component is faulty. A faulty path component is assumed to permanently malfunction by either corrupting the packet payload or causing the packet itself to be miss-routed and/or dropped.

9.6.1.2 Network Test Concept

The task of network fault diagnosis is inherently challenging due to the fact that the path components under test are deeply embedded within a complex interconnection structure and consequently their state cannot be directly observed. However, an examination of the path function $P(C)$ truth table, shown in Fig. 9.8c, suggests a promising approach. Although there are 2^n-1 possible fault configurations (where n is the number of components for some arbitrary route) satisfying $P(C)=0$, there is only a single state that satisfies $P(C)=1$ which is when $C_i=1$ for all *i*. The practical application of this is that whenever we can verify that a test packet has been successfully routed over some specified network path without being corrupted, it is assumed that the underlying components comprising that path are fault-free. So by routing packets through the network such that every component is part of at least one good path it is ultimately possible to render a complete network diagnosis on the *assumed faulty until proven good* basis.

To produce this outcome, we have proposed a route generation method based on an adaptation of the self-avoiding walk (SAW) [22]. The main idea is that random-like paths are computed at a source subject to the requirement that a packet may not pass through a given path component more than once for a particular route.

Note that this restriction does not necessarily preclude multiple traversals via the same router. In addition, the following constraints are applied:

- Explicitly defined start and end points;
- The physical boundary of the network topology;
- A maximum specified number of router hops.

The element of randomness afforded by SAW type routes proves to be an effective means for path discovery in the presence of unknown arbitrarily distributed faults. It is also this characteristic that makes the proposed approach highly amenable to arbitrary network topologies.

9.6.1.3 Diagnosing the NoC

Since the GPD is not embedded within the RFD(s) under test, the first phase of network diagnosis begins by testing the inter-die connectivity between the GPD and the RFD to which it is directly connected. This is accomplished by the GPD injecting one or more test packets into the RFD along a SAW type path that returns back to the GPD. Successful reception of a previously injected packet verifies the correctness of that link and enables further testing of the RFD along the fault-free path.

For testing the RFD, the previously described method is modified into a series of operations wherein the GPD writes to a designated core along one path followed by a read from that same core along a different path, both based on a SAW. Terminal ports are selected systematically in order to cover all network-to-core connections. Router-to-router connections are exercised in a random fashion as a consequence of the SAW path generation for the targeted read/write operations. Each successful read operation establishes the fault-free status of the path components along both the read and write paths used, as well as the associated network interface. Testing of RFDs indirectly linked to the GPD via one or more of the other RFDs requires additional tests aimed at verifying the usability of all inter-RFD links. The final diagnosis result includes the faults from every testable path component.

9.6.2 Xentium Tile Dependability

The dependability approach used for the Xentium tiles in the RFD adopts several key concepts from prior research to enhance the dependability of homogeneous multi-processor architectures, including the "Know-Good-Die/Tile" concept, and concepts related to test reuse and majority-voting among identical processing tiles [23]. When we apply identical test stimuli to fault-free tiles-under-test, we expect to obtain identical test responses from these as they are all identical. If at least three tiles are tested in parallel, we can identify a faulty tile from the differences in its test responses

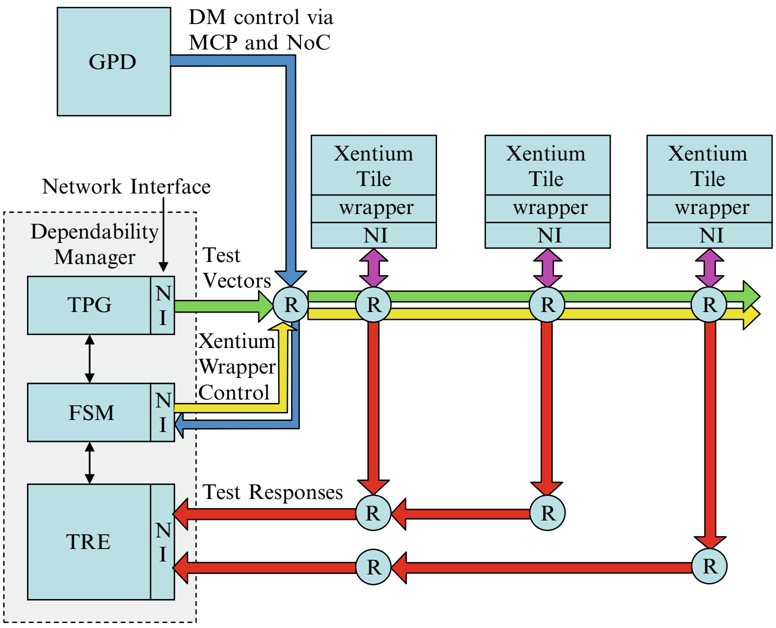


Fig. 9.9 GSP dependability architecture for the Xentium tiles

compared to the others, as long as no more than a minority of tiles-under-test (in this example one tile out of three tiles tested) becomes faulty at a time.

To permit periodic testing of tiles *at run-time* this method however requires additional, on-chip infrastructural IP that has to be designed and integrated in the RFD architecture at design-time. Our RFD dependability architecture includes a dependability wrapper around each Xentium tile, and a Dependability Manager on each chip [24]. The wrapped tiles are connected to the NoC in the same way as the original tiles were.

During dependability testing, the NoC is used as a test access mechanism to transport test stimuli to and test responses from the wrappers of the tiles-under-test. The NoC makes no difference between the transportation of functional data and dependability test data. Since the total bandwidth of the NoC therefore has to be shared between the application's functional data and the dependability infrastructure's test data, dynamically pausing and resuming of the dependability test is supported and used to ensure that the required NoC bandwidth is always available for the running applications. This *prioritization* of functional data over dependability test data ensures that the dependability tests do not impact the performance of the user application(s).

Figure 9.9 shows the RFD dependability architecture for the Xentium tiles, including its most important components: the Dependability Manager (DM), the Xentium tile wrappers around the Xentium tiles, and the GPD executing the Xentium tile dependability software.

9.6.2.1 Dependability Manager

The DM shown in Fig. 9.9 consists of a test pattern generator (TPG), a test responses evaluator (TRE) and a finite state machine (FSM) [25, 26]. The FSM controls the DM and communicates with the dependability software running on the GPD. Deterministic test patterns for the Xentium tile(s) (with 32 parallel scan-chains) have been generated at design-time using a commercially available tool. The TPG in the DM reproduces these deterministic test patterns using a linear-feedback shift register combined with a reseeding technique.

A phase-shifter packs the test stimuli into 32-bit words to suit the 32-bit wide NoC. Finally, each bit of such a 32-bit word fills one scan flip-flop in each of the 32 scan chains inside the Xentium tile. The GPD, DM and Xentium Tiles are connected to the NoC through NoC-routers (R). The silicon area of the DM is approximately one percent of the total area of the RFD.

9.6.2.2 Xentium Tile Wrapper

The Xentium Tile Wrapper allows switching the operating mode of the Xentium tile between functional mode and dependability test mode via commands issued by the DM over the NoC [24]. In functional mode, the wrapper transparently delivers data from the NoC to the functional inputs of the Xentium tile, and passes data from the functional outputs to the NoC. In dependability test mode, it delivers data from the NoC to the test inputs (scan-chain inputs and primary inputs) of the tile. A similar operation is performed at the output of the tile where scan-chain outputs and primary outputs are captured and passed to the NoC by the wrapper.

9.6.2.3 Tile Processor Dependability Software

The tile processor dependability software is an essential part of the dependability enhancement approach for the GSP. This software executes on the GPD, where it starts the background test activities. It requests sufficient RFD resources (i.e. NoC bandwidth and Xentium tiles) to use the DM together with two or three available Xentium tiles. The run-time resource manager subsequently allocates and configures the required communication routes between these components. The DM itself does not require knowledge of which cores it is testing. This is completely determined by the routes that are configured in the NoC.

The test sequence is started when the GPD writes into an FSM control register in the DM. The DM subsequently switches the wrappers of the chosen Xentium tiles to dependability test mode, preparing the Xentium tiles for receiving the test stimuli generated by the TPG of the DM. The test stimuli are subsequently multicasted via the NoC to the target Xentium tiles, and the test responses are collected and compared in the TRE of the DM.

This test sequence is halted as soon as a difference is detected in the test responses from the Xentium tiles under test, or as soon as the test is completed. The Xentium tile that generated a test response that differs from the others is identified as the faulty tile. This information is encoded into a fault status report that can be retrieved by the dependability software executing on the GPD. The faulty Xentium tile can then be isolated or removed from the usable resource table of the run-time resource management software. If no test-response differences are detected during the dependability test, all tested tiles are considered fault-free, and made available to the run-time resource management software.

This dependability test process is repeated until all Xentium tiles are tested. This way, the faulty tiles, if any, are identified and isolated from the system ensuring that the remaining Xentium processing tiles are fault-free. Depending on the dependability requirements (i.e. acceptable mean system down time / unavailability) from the end user, the dependability test activities can be performed at a desired frequency.

9.7 Preliminary Experimental Results

Reaching the ambitious objectives of the CRISP project involves software development, digital hardware design and manufacturing of two ICs and one Printed Circuit Board (PCB). The key result is prototyping the GSP platform, consisting of 45 Xentium DSP cores with run-time resource management and dependability support running both high-end and low-end streaming DSP applications. At the time of writing, the project is still running and, therefore, only preliminary results are described here. We describe, first, the results of the hardware manufacturing and, second, how the applications will be demonstrated on the GSP prototype. Demonstration of the applications on the manufactured GSP hardware unites all topics (i.e. reconfigurable multi-core SoC, low-end and high-end streaming applications, run-time resource management, and dependability) of the CRISP project.

9.7.1 Hardware Manufacturing

The CRISP project implements a pragmatic research and engineering schedule to research, conceive, implement, and manufacture two ICs and a PCB. At the point of writing, the PCB and the GPD chip are ready and the RFD dies have just left the foundry.

The PCB integrates five RFD chips and one GPD chip on a single board and prototypes the instance of the GSP architecture illustrated in Fig. 9.3, with 45 Xentium cores and one ARM® core. On the PCB, the NoCs for the RFDs are interconnected via the Multi Channel Port (MCP) interfaces. Additionally, the PCB serves as a verification platform for implementing demonstrators and integrates various I/O

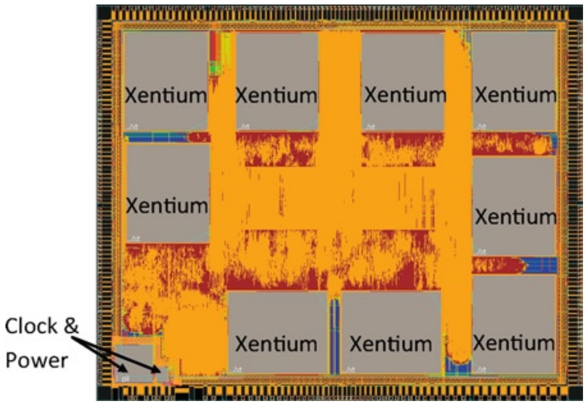


Fig. 9.10 Layout of reconfigurable fabric device (RFD) chip

interfaces and an FPGA. The FPGA is connected to all RFDs via MCP interfaces. Hence, direct access to the NoC of every RFD is provided via the FPGA.

The GPD chip is manufactured in UMC 130 nm CMOS Technology, runs at 200 MHz, and is delivered in a 400-pin BGA package. The supply voltage is 1.2V core and 3.3V I/Os. The estimated power dissipation is 400mW.

The RFD chip is manufactured in UMC 90 nm CMOS Technology. The die size is 44 mm^2 with 344 kBytes memory on-chip. The RFD will run at 200 MHz and will be delivered in a 400-pin BGA package. The supply voltage is 1.0V core and 3.3V I/Os. To manage the complexity of the RFD design, a hardmacro was created for the Xentium Tile Processor Cores. The RFD integrates nine Xentium hardmacros.

Figure 9.10 gives an overview of the chip layout. The nine big grey blocks are Hardmacros of the Xentium Processing Tile. Two Memory Tiles are located in the middle and clock and power blocks are found in the lower left corner. The logic of the dependability manager and NoC is laid out between the processing and memory tiles.

9.7.2 Beamforming Demonstrator

The CRISP beamforming demonstrator has 16 receive channels and computes 8 beams. Figure 9.11 depicts the architecture of the beamformer on the General Streaming Processor (GSP) platform. The demonstrator uses all five RFDs. Four RFDs are used for the signal processing of the 16 receive channels; each of these RFDs does antenna processing for four receive channels at an input rate of 1.28 Gbps. The beam forming function is implemented on the fifth RFD, combining the data from the 16 processed receive channels to compute eight beams at an output rate of 640 Mbps. The ongoing implementation maps the processing tasks onto 39 of the 45 DSP cores.

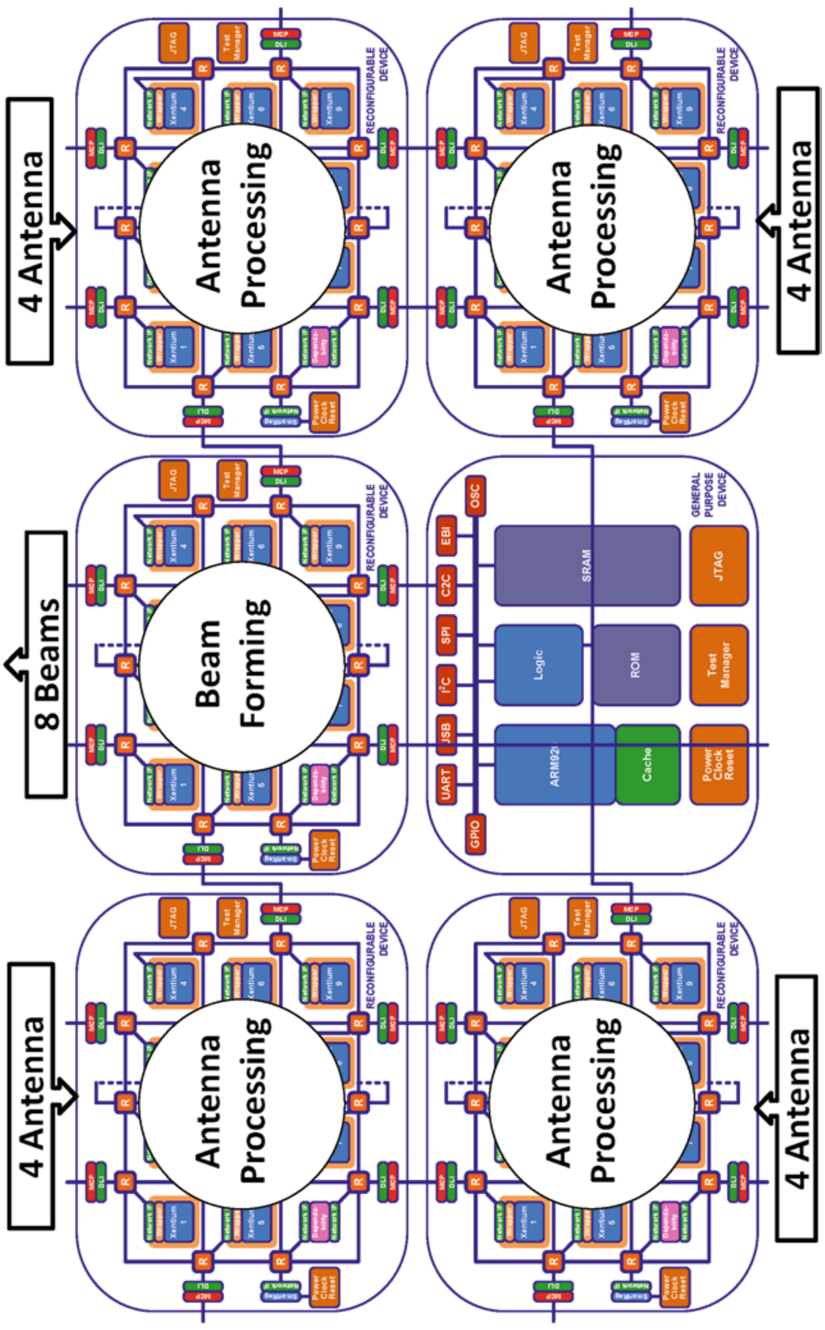


Fig. 9.11 Illustration of a mapping onto the GSP platform of the 16 channel / 8 beams digital beamforming demonstrator. Four RFDs are used for antenna processing; each processing the input of four channels at an input rate of 1.28 Gbps. One RFD is used to form eight beams from the sixteen channels at an output rate of 640 Mbps

9.7.3 GNSS Demonstrator

The GNSS application is demonstrated on the CRISP verification platform containing five RFDs using a commercial off-the-shelf radio front end. The resulting digital data stream received by the front end is fed to the verification platform and inserted in the RFD using the MCP interface.

The tasks of the GNSS application are illustrated in Fig. 9.2. All digital baseband processing can be done on one RFD. The parallel digital signal processing tasks are assigned to multiple Xentium processing tiles in the RFD and include acquisition (search of satellites using long FFTs) and tracking (data decoding and signal time-of-arrival measurements by means of serial correlation). This part of the application benefits from the parallel nature of RFD. The parallel tasks are preceded by a single task that preprocesses the input signal to ease the computational burden of the parallel tasks.

Figure 9.12 illustrates an example mapping of the GNSS application to the RFD and GPD. Three kinds of processes are mapped onto the RFD. Input preprocessing is closest to the inserting point of the input stream coming from the GNSS RF front end through the FPGA. Two acquisition processes are executed in the upper part of the RFD and four processes tracking satellite signals are executed on the right half. The outcome of the acquisition and tracking is forwarded to a single task of navigation, which is performed on the GPD. GPD forwards the navigation solution to an external PC via a serial connection. The implementation steps towards the CRISP GNSS application are explained in more detail in [27, 28]. The GNSS application is able to solve the position, velocity, and time of the receiver when four or more satellites are tracked successfully [12]. Thus, the receiver application should always (after an initial acquisition stage) have four or more cores running a tracking process to enable navigation.

9.7.4 Enabling Graceful Degradation by Reconfiguration

Dynamic resource management allows for fault-tolerance in the case that hardware faults can be circumvented, using a disjoint set of resources. However, the amount of resources required by the application does not change. In case that the platform is either seriously compromised, or its available resources are near depletion, applications may be refrained from execution. For robustness reasons, even more flexibility may thus be added to an application. By providing multiple quality-of-service levels the probability may increase that an application is allowed to start, albeit in a reduced form. This scenario is already demonstrated within the CRISP project with the GNSS application.

The next challenge is to seamlessly switch between the quality-of-service levels provided by an application. Scenarios exist where it is preferred to scale down running applications to allow additional functionality to be performed on the same

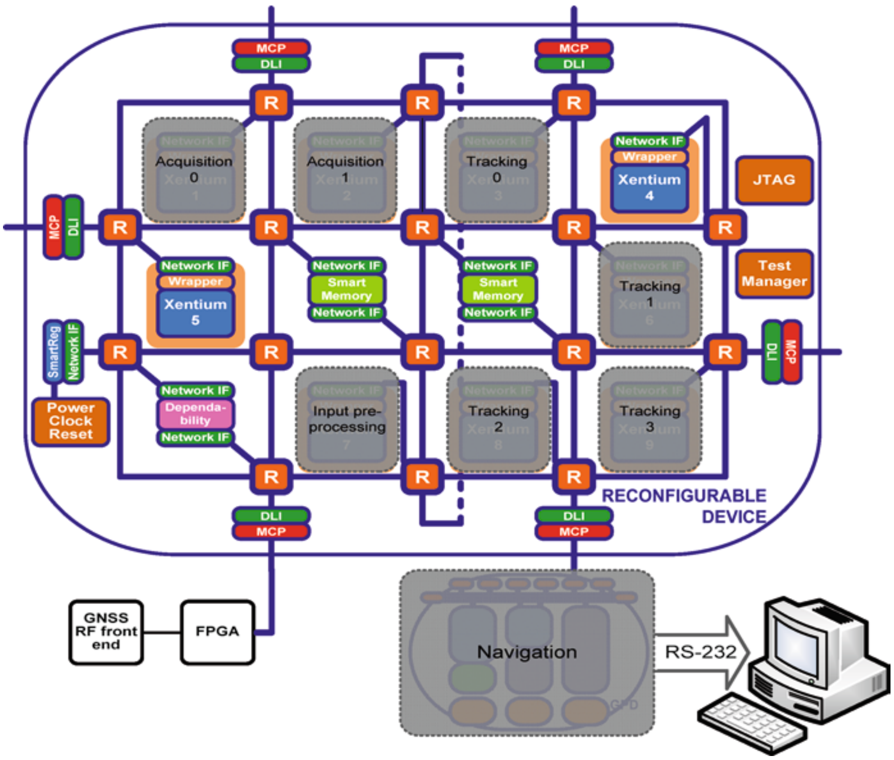


Fig. 9.12 Illustration of mapping of GNSS tasks onto the GSP using seven Xentiums on one RFD

platform. In other scenarios, hardware faults may trigger a reconfiguration of an application, such that it gracefully degrades by running in a reduced mode on a (slightly) different set of resources.

9.8 Outlook

New projects such as STARS [29] and NEST [30] have been launched to continue research along the path set out by CRISP and to pick up on CRISP results. Furthermore, CRISP motivations and goals remain highly relevant. It is expected that conventional computing architectures will be replaced by more flexible reconfigurable multi-core computing platforms and streaming applications are expected to create a huge drive and momentum for this shift.

The CRISP project anticipates on this expected architectural shift by researching streaming applications, reconfigurable IP cores, interconnect technologies, run-time tools and dependability issues. Europe is a major player in the embedded arena today and this holistic approach is necessary for European companies to achieve world leading positions in computing solutions and products for streaming applications.

CRISP aims at the important domain of streaming applications, which is expected to grow faster than other application domains with the adoption of new standards. The CRISP project will deliver a novel reconfigurable multi-core computing platform. These types of platforms are urgently needed, since hardwired logic (i.e. ASIC) is getting prohibitively expensive for an ever wider range of products. Currently, many products make use of fine-grained reconfigurable FPGAs. Their usage, however, results in severe overheads unacceptable for most mass market applications. These overheads are related to device area, power consumption, and unit costs.

Embedded systems increasingly profit from the integration of signal processing capabilities of new energy-efficient multi-core architectures with added flexibility. By combining techniques from application domain and computer engineering, embedded systems can improve the production of solutions both concerning costs and time:

Reduction of costs – by reusing IP building blocks in SoC designs, the development and verification costs of a design cycle will be reduced and design productivity be increased. Ultimately, reconfigurable multi-core platform chips can be used for a large variety of applications by reconfiguring the same chip after fabrication;

Reduction of time to market – by using energy-efficient general-purpose multi-core architectures as proposed in CRISP and abstracting from the application-specific platform, the development time of new solutions can be drastically reduced.

Moreover, the ability to reconfigure a system and to use this capability to make systems dependable is extending product lifetime. Because users can reconfigure their purchased product to increase service levels, hardware can be used for a longer period of time. Furthermore, built-in dependability techniques allow for graceful degradation. Because of these aspects, products will have a longer life cycle, become cheaper, and be less of a burden to the environment.

Dependable and reconfigurable multi-core systems will serve various application domains ranging from consumer electronics to mission critical space applications. For instance, in space applications, extending lifetime under harsh conditions, and easy upgrading are important aspects when designing dependable and reconfigurable systems. However, for consumer electronics faster time-to-market is a differentiator to win the competition. Both orthogonal application domains focus on improving dependability in combination with reconfigurable multi-core hardware, however, motivated by partly different grounds.

9.9 Conclusions

The holistic and pragmatic approach of CRISP includes hardware design and manufacturing as well as development of application, system, and development software. This approach enables research into fundamental trade-offs in the intersection of hardware and software where reconfigurable computing resides. The scalable NoC-based many-core General Stream Processor (GSP) architecture developed in CRISP delivers sufficient and scalable stream processing performance for a wide range of applications in both high- and low-end markets. The GSP is equipped with a sophisticated run-time resource management system that can dynamically change the allocation of application tasks to processing cores and data transfers to communication channels. The run-time resource management turns the GSP into a flexible and efficiently programmable coarse-grained reconfigurable computing platform. Moreover, CRISP combines the run-time management with dedicated dependability hardware and software to create a self-repairing dependable GSP. The approach is a novel way to exploit the inherent redundancy of NoC-based many-cores and to address tomorrow's predicted issues with accelerated degradation of ICs as processing geometries continue to shrink.

Acknowledgments CRISP is a multi-disciplinary endeavor. To achieve the project objectives, highly specialized knowledge is required about applications, front-end and back-end hardware design as well as development and system software. The work reported in this chapter is a team effort and the authors would like to thank the many experts from the consortium, who worked on CRISP including Sébastien Baillou, Helmut Baur, Arjan Boeijink, Werner Brugger, Christoph Büchler, Jan de Wilde, Jeroen Flierman, Michael Günther, Matthias Hofinger, Klaas Hofstra, Henk Holdijk, Philip Hölzenspies, Thomas Janz, Erik Karstens, Ben Kemper, Roy Krikke, Pieter Maier, Rene Meerman, Rik Portengen, Jordy Potman, Johannes Schäfer, Jeroen Scholten, Hans Schurer, Lodewijk Smit, Hermen Toersche, Rene van Hees, and Mark Westmijze. Further, the authors would like to thank the European Community's Seventh Framework Programme FP7/2007-2013 for supporting the CRISP project, and the project officers and project reviewers for their support and help in guiding the project towards its objectives.

References

1. Dally W. et al. Stream Processors: Programmability with Efficiency, ACM Queue, pp. 52–62, 2004.
2. U.J. Kapasi, S.R. Rixner, W.J. Dally, B. Khailany, J.H. Ahn, P. Mattson, and J.D. Owens. Programmable stream processors. In IEEE Computer 36(8), pp 54–62, 2003.
3. International Technology Roadmap for Semiconductors (ITRS), http://www.itrs.net.
4. 4S Project Chapter of this book.
5. 4S Project web pages http://www.recoresystems.com/research/.
6. G.Smit, E. Schuler, J. Becker, J. Quevremont, and W. Brugger: Overview of the 4S project. In *Proc.* 7th International Symposium on System-on-Chip (SoC'05), 2005.
7. M.J.G. Bekooij et al. Dataflow Analysis for Real-Time Embedded Multiprocessor System Design. In Dynamic and Robust Streaming between Connected CE Devices, Kluwer, 2005.
8. H. J. Visser, Array and phased array antenna basics. Chichester, West Sussex, UK: Wiley, Sep. 2005.
9. M. I. Skolnik, Introduction to Radar Systems, 3 rd ed. New York, NY, USA: McGraw-Hill, 2001.
10. GPS Interface Control Document (ICD-GPS-200D), IRN-200 C-004, U.S. Air Force, 2004.
11. "Galileo Open Service, Signal In Space Interface Control Document (OS SIS ICD)," European GNSS Supervisory Authority, draft 1, 2008.

12. M. Braasch and A. J. Van Dierendonck. GPS Receiver Architectures and Measurements. In Proc. of the IEEE, vol. 87, no. 1, pp. 48–64, 1999.
13. D. Akos. The role of Global Navigation Satellite System (GNSS) software radios in embedded systems. In GPS Solutions, 2003.
14. S.K. Eo, S. Yoo, K.M. Choi. An industrial perspective of power-aware reliable SoC design. In Proc. Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 555–557. IEEE Computer Society Press, 2008.
15. J. Kurzak, A. Buttari, P. Luszczek, and J. Dongarra. The Playstation 3 for High-Performance Scientific Computing. In Computing in Science & Engineering, Vol. 10, Issue 3, IEEE Computer Society, 2008.
16. J. Dongarra, A. Lastovetsky. An overview of heterogeneous high performance and grid computing. In Engineering the Grid: Status and Perspective, 2006.
17. O.M. Moreira, M.J.G. Bekooij. Self-timed scheduling analysis for real-time applications. In: EURASIP Journal on Advances in Signal Processing, vol. 2007, pp. 24–37, 2007.
18. M.L. Fisher, R. Jaikumar, and L.N.V. Wassenhove. A multiplier adjustment method for the generalized assignment problem. In Management Science 32(9), pp 1095–1103, 1986.
19. T.D. ter Braak, P. K.F. Hölzenspies, J. Kuper, J. L. Hurink, and G. J.M. Smit. Run-time Spatial Resource Management for Real-Time Applications on Heterogeneous MPSoCs. In Proc. Conference on Design, Automation and Test in Europe (DATE), pp. 357–362, 2010.
20. T.D. ter Braak, S.T. Burgess, H. Hurskainen, H.G. Kerkhoff, B. Vermeulen, X. Zhang: On-Line Dependability Enhancement of Multiprocessor SoCs by Resource Management. In Proc.12th International Symposium on System-on-Chip (SoC'10), p103–110, 2010.
21. S. Burgess, T. Ahonen, and J. Nurmi: Software Based Approach to Fault Diagnosis for Multi-Die Networks-on-Chip. In Proc. System, Software, SoC and Silicon Debug, 2010.
22. B. Hayes. How to avoid yourself. In *American Scientist*, vol. 86, no. 4, 1998.
23. H.G. Kerkhoff, O. Kuiken, and X. Zhang: Increasing SoC Dependability via Known Good Tile NoC Testing. In Proc. Conf. on Dependable Systems and Networks (DSN'08), 2008.
24. X. Zhang, H.G. Kerkhoff, B. Vermeulen. On-Chip Scan-Based Test Strategy for a Dependable Many-Core Processor Using a NoC as a Test Access Mechanism. In Proc. 13th Euromicro Conference on Digital System Design (DSD), 2010.
25. O.J. Kuiken, X. Zhang and H.G. Kerkhoff. Built-In Self-Diagnostics for a NoC-Based Reconfigurable IC for Dependable Beamforming Applications. In Proc. IEEE Intern. Symp. on Defect and Fault Tolerance in VLSI Systems (DFT'08), 2008.
26. H.G. Kerkhoff and X. Zhang. Design of an Infrastructural IP Dependability Manager for a Dependable Reconfigurable Many-Core Processor. In Proc. DELTA'10, 2010.
27. H. Hurskainen, J. Raasakka, T. Ahonen, and J. Nurmi. Multicore Software-Defined Radio Architecture for GNSS Receiver Signal Processing. In EURASIP Journal on Embedded Systems, vol. 2009, Article ID 543720, 10 pages, 2009.
28. J. Raasakka, H. Hurskainen, T. Paakki, and J. Nurmi. Modeling Multi-Core Software GNSS Receiver with Real Time SW Receiver. In Proc. ION GNSS, 2009.
29. Sensor Technology Applied in Reconfigurable Systems for sustainable Security (STARS) Project, National Dutch Project, http://www.starsproject.nl.
30. Nederland Streaming (NEST) Project, Dutch Technology Foundation STW Project, http://caes.ewi.utwente.nl/caes/index.php/research/recently-started-projects/nest