Communicating Process Architectures – 2003
Jan F. Broenink and Gerald H. Hilderink (Eds.)
IOS Press, 2003

Parallel Processing — the picoChip way!

Andrew DULLER Gajinder PANESAR Daniel TOWNER
picoChip Designs Ltd., Bath, UK

Abstract. This paper describes a new approach to parallel processing within the well targetted application domain of wireless communications systems, using the picoArray™. The picoArray™ is a tiled-processor architecture, containing 430 heterogeneous processors, connected through a novel, compile-time scheduled interconnect. We show how the features of the picoArray™ allow deterministic processing to be achieved, and how the tool chain allows programming to be performed effectively in a combination of high level assembly language and C. By handling a wide variety of types of processing within the picoArray™ a single design flow can be used to produce complex communications systems. The effectiveness of this approach is demonstrated through the use of the picoArray™ to build a working 3G base-station.

1 Introduction

Within the applications space of wireless communications, systems are typically designed using a mixture of DSPs, FPGAs and custom ASICs, resulting in systems that are awkwardly parallel in nature. In addition, the tool chain forces the user to regard the systems from several different perspectives, introducing problems with specification, implementation and verification. All these factors result in solutions that are complex, non-deterministic, and difficult to debug.

The picoArray™ architecture [1] was developed to create a massively parallel system which did not suffer from many of the problems of conventional general purpose parallel systems. There was no intention to create a general purpose parallel processor but one which provided an alternative to creating an ASIC in a range of applications within the wireless communications domain. In some senses it can be thought of as a "programmable ASIC". The fundamental attributes of the approach are that communications between processes have to be fixed at compile time and therefore no support for dynamic communications is required and problem partitioning is largely done by the user since this is what would occur during an ASIC design process.

The design of both the picoArray™ and the tool chain strongly reflect the application domain whose attributes are:

- application partitioning is traditionally done by the user and is well understood,
- high degree of replicated code due to multi-user and multi-antenna systems,
- considerable use of stream based processing [1],
- hand coding of assembly language is often used to achieve compactness of code and efficiency.

[1] Stream based processing is where data is passed at high rate through a chain of processes each of which do relatively simple operations on the data before passing them on to the next stage.

This paper consists of an overview of the picoArray™ architecture, an introduction to the applications domain and a summary of the tool chain. Finally, the current status of the work is described which helps to show the effectiveness of the approach through the goals that have been achieved.

2 Overview of the picoArray™ architecture

The picoArray™ is a tiled processor architecture in which 430 heterogeneous processors are connected together using a deterministic interconnect as shown in figure 1. The interconnect consists of bus switches joined by picoBus™ connections. To simplify the diagram the interconnect is only drawn between the switches, each processor is actually connected directly to the picoBus™ above and below it (an enlarged view of part of the interconnect is shown in figure 2, again to simplify the diagram only two of the four vertical bus connections are shown).

The level of parallelism is relatively fine grained with each processor having a small amount of local memory. There are four RISC processor variants which share a common core instruction set, but have varying amounts of memory and additional instructions to implement certain wireless baseband control and digital signal processing functions. Each processor runs a single process in its own memory space and they use "signals" to synchronise and communicate. Multiple picoArray™ devices may be connected together to form systems containing thousands of processors using on-chip peripherals which effectively extend the on-chip bus structure. A brief description of the four processor variants and a breakdown of the internal memory distribution is given in table 1.

The initial version of the picoArray™, the PC101, is a large chip and it is therefore necessary to use redundancy within the array to improve the yield.

The routing strategy used was determined largely by the real time nature of the intended applications where the indeterminate latency due to bus arbitration would be unacceptable. All of the communications are determined during the "compilation" of the system which means that the communications bandwidth can be guaranteed.

2.1 Interconnect

Within the picoArray™ , processors are organised in a two dimensional grid, and communicate over a network of 32-bit unidirectional buses (the picoBus™) and programmable bus switches. The physical interconnect structure is shown in figure 2. The processors are connected to the picoBus™ by ports which contain internal buffering for signal data. These act as nodes on the picoBus™ and provide a simple processor interface to the bus based on *put* and *get* commands. The processors are essentially independent of the ports unless they specifically use a *put* or a *get* instruction.

The inter-processor communication protocol implemented by the picoBus™ is based on a time division multiplexing (TDM) scheme. There is no run-time bus arbitration, so communication bandwidth is guaranteed. Data transfers between processor ports occur during specific time slots, scheduled in software, and controlled using the bus switches. Figure 2 shows an example in which the switches have been set to form two different signals between processors. Signals may be point-to-point, or point-to-multi-point. In the latter case, the data transfer will not take place until all the processor ports involved in the transfer are ready.

Communication time slots throughout the picoBus™ architecture are allocated according to the bandwidth required. Faster signals are allocated time-slots more frequently than slower signals. The user specifies the required bandwidth for a signal by giving a rate at which the

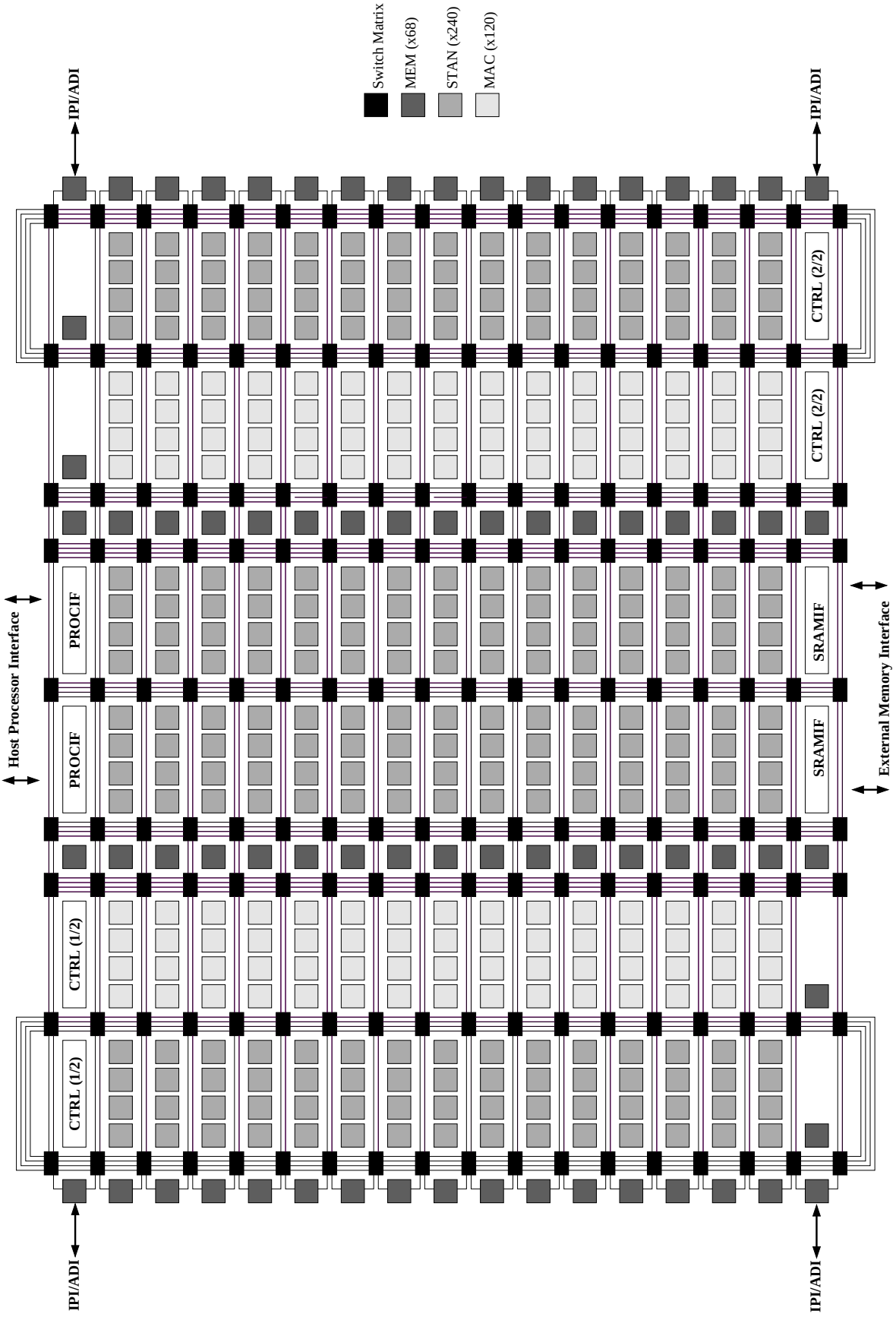

Figure 1: Top-level Diagram showing processors and interconnect

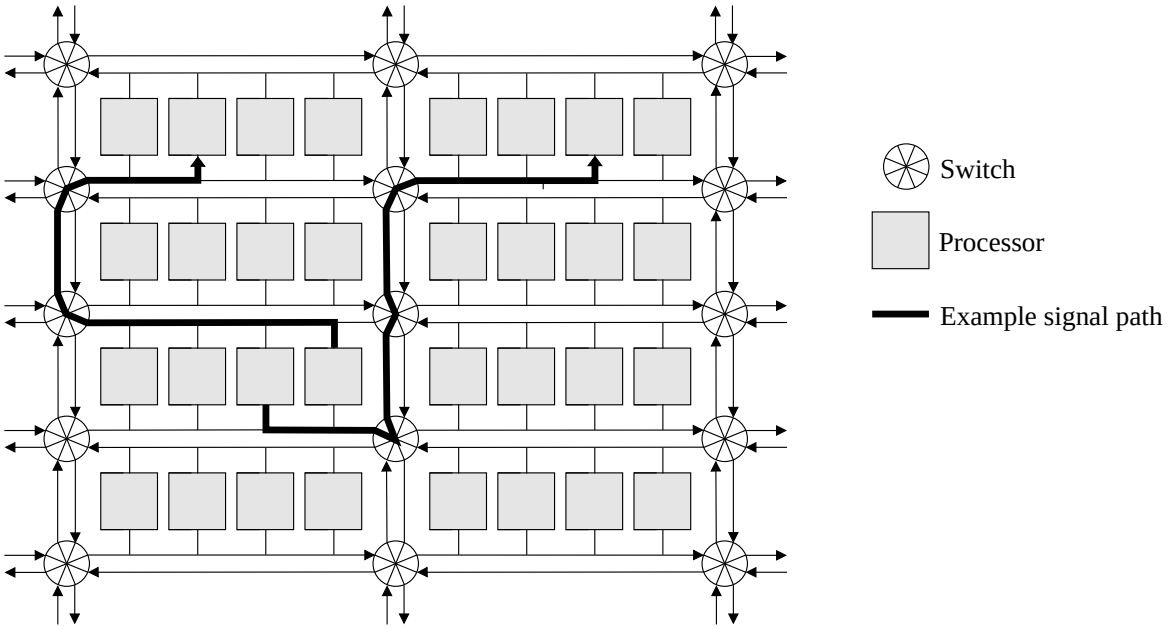

Figure 2: Interconnect

signal must communicate data. For example, a transfer rate might be described as @4, which means that every fourth time-slot has been allocated to that transfer.

The default signal transfer mode is synchronous; data is not transfered until both the sender and receiver ports are ready for the transfer. If either is ready before the other then the transfer will be retried during the next available time slot. If, during a *put* instruction no buffer space is available then the processor will sleep (hence reducing power consumption) until space becomes available. In the same way, if during a *get* instruction there is no data available in the buffers then the processor will also sleep. Using this protocol ensures that no data can be lost.

There is also an asynchronous signal mode where transfer of data is not handshaken and in consequence data can be lost by being overwritten in the buffers without being read.

2.2 Processors

All of the processors in the picoArray™ are 16-bit, and use 3-way VLIW scheduling. The basic structure of the processors is shown in figure 3. Each processor has its own small memory (between 1KB and 32KB), which is organised as separate data and instruction banks (i.e. a Harvard architecture). The processor contains a number of communication ports, which allow access to the interconnect buses through which it can communicate with other processors. Each processor is programmed and initialised using a special configuration bus. The processors have a very short pipeline which helps programming, particularly at the assembly language level. The architecture of the four processor variants are shown in figure 4.

In addition to the general purpose processors, there are a number of special peripherals, including a host interface, an SRAM interface, asynchronous data and inter-picoArray interfaces. These peripherals are connected to the bus structure through ports, which enables them to be treated as though they are special purpose processes. The overall distribution of processors and peripherals is shown in figure 1 with the peripherals being placed in the top and bottom rows of the array.

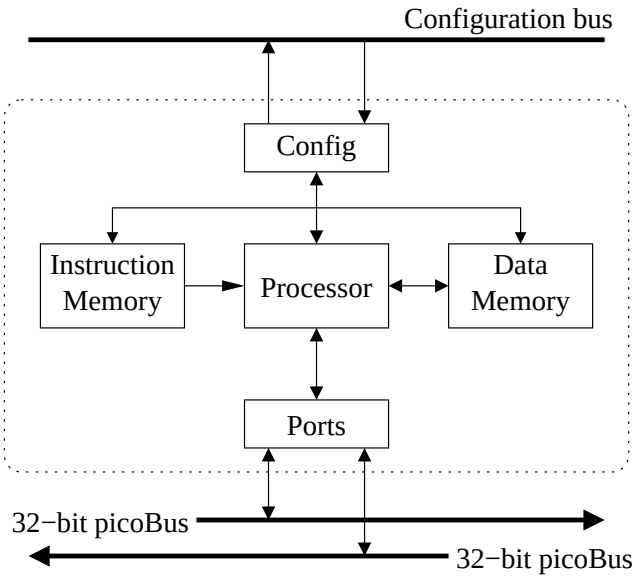

Figure 3: Processor Structure

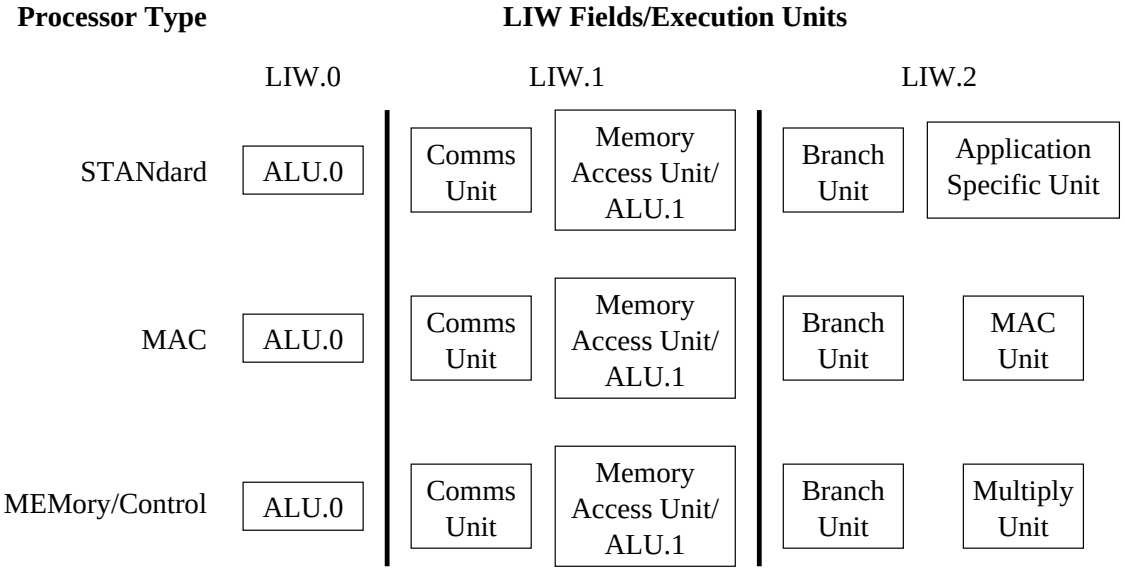

Figure 4: VLIW and execution unit structure in each processor

Type	Description	Number	Memory (bytes)
STAN	*Standard*: A standard processor optimized for CDMA spread and de-spread and other wireless base station signal processing functions	240	768
MAC	*Multiply accumulate*: A processor that includes a multiply-accumulate unit and which supports additional multiply-accumulate instructions	120	768
MEM	*Memory*: A processor having a multiply unit and additional data memory	68	8,704
CTRL	*Control*: A processor with a multiply unit and larger amounts of data and instruction memory optimized for the implementation of control functionality	2	32,768

Table 1: PC101 processor variants and memory distribution

2.3 Host interface

The Host or microprocessor interface is used to configure the picoArray™ device and to transfer data to and from the picoArray™ device using either a register transfer method or a DMA mechanism. The DMA memory-mapped interface has a number of ports mapped into the external microprocessor memory area. Two ports are connected to the configuration bus within the PC101 and the others are connected to the internal picoBus™. These enable the external microprocessor to communicate with the internal array elements using signals.

2.4 SRAM interface

Each picoArray™ has an amount of memory distributed amongst the processors for data and instruction storage. However, an external SRAM interface is provided to supplement the on-chip memory. This interface allows processors within the core of the picoArray™ to access external SRAM across the internal picoBus™.

2.5 Asynchronous data/Inter-picoArray interfaces

There are four interfaces on each device which can be configured in one of two modes: either the inter picoArray interface (IPI) mode or the asynchronous data interface (ADI) mode. The choice of interface mode is made for each interface separately during device configuration.

2.5.1 Inter picoArray interface

The four IPI interfaces are bidirectional and designed to allow each picoArray™ to exchange data with up to four others. Using this feature, a grid of picoArray™ devices can be constructed to implement highly complex and computationally intensive signal processing systems. The IPI interface operates in full duplex, sending and receiving 32-bit words. The 32-bit words on the internal picoBus™ are multiplexed as two 16-bit data on the interface itself.

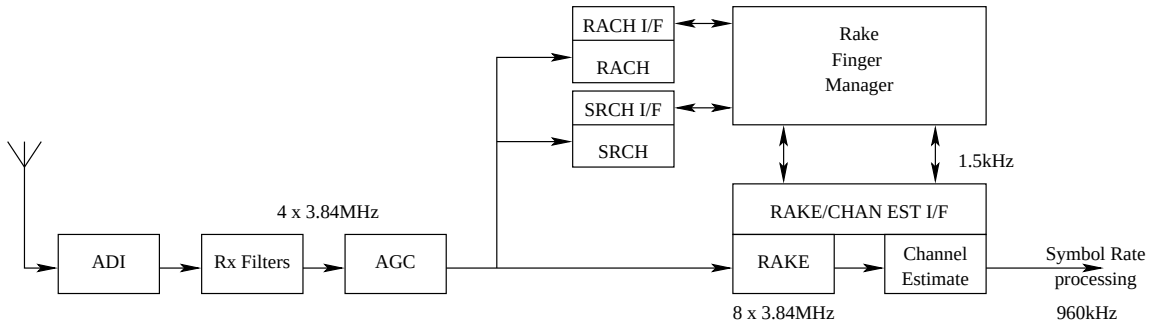

Figure 5: Typical data rates

2.5.2 Asynchronous data interface

The asynchronous data interface (ADI) allows data to be exchanged between the internal picoBus™ and external asynchronous data streams such as those input and output by data converters or control signals between the base band processor and the RF section of a wireless base station.

2.6 Low-power considerations

Potentially, a device such as the PC101, which contains 430 processors and a TDM interconnect, could use a lot of power. A number of methods have been used to enable the power consumption of the picoArray™ to be reduced. For example, individual processors are able to 'sleep' when they are waiting for communications events, thus consuming minimal power, and parts of the picoArray™ which are not used in a particular design are switched off.

2.7 Array layout

As can be seen from section 3 the target applications are fairly varied although they have many common attributes. The layout of the array in terms of quantities and locations of processors was determined to match these attributes as far as possible. Since any realistic system will make use of many picoArray™ devices, 4, 8, 16 or more, the layout has to be a compromise between the different types of processing that are required within a wireless infrastructure system. At the input to the system the data rates are very high but the processing is simple, as data flows through the system the data rates reduce, the control becomes more complex and the operations become more complex.

Figure 5 shows a typical piece of processing from the front end of the receive chain in a base station. At the input to the system, through the ADI, the data rate is high during which the incoming signals are filtered. This is then transformed into a symbol rate stream of data at a much lower rate, this rate can vary between the 960kHz given, down to 15kHz depending on the type of signal. It should be noted that this rate is for each user of the base station and typically there will be 64 users or more. In addition, multiple antennas will be used making the initial input data rate even higher.

The majority of the array, 360 processors (STAN and MAC processors), are designed for stream based processing and therefore have small amounts of local memory. The split between STAN and MAC was determined largely by the requirement for the specialist spread and de-spread operations (present in the STAN processor) required by CDMA. The target applications tend to have a smaller requirement for block based processing and this is supported by the 68 MEM processors which have more local storage and can be used in conjunction

with the SRAM interface. The low level control requirements of application systems are handled by the provision of 2 CTRL processors. Each processor has a performance comparable to an ARM 9 for control type functionality, or a TI C55XX for DSP tasks.

3 Target applications

The picoArray™ is targetted at providing the processing solution for a wide range of communications systems, specifically those related to wireless infrastructure. These include the UMTS FDD/TD-SCDMA [2], IEEE 802.16 [3] and CDMA2000 [4] standards. While the details of these standards vary considerably the style of processing and algorithmic requirements have many similarities. The typical functions are:

- Signal filtering and conditioning for both transmit and receive paths (adjacent channel rejection, digital pulse shaping, Inter-Symbol-Interference minimisation)
- Multiple access processing: including both mux and demux, and its management (CDMA/OFDMA/TDMA)
- Signal synchronisation (timing acquisition and tracking and local oscillator offset compensation)
- Equalisation: compensation for channel distortion (terrestrial mobile radio channel is probably the most severe of any communications channel)
- Forward error correction: Encode and decode. CRC, Maximum Likelihood Viterbi and Turbo (the many flavours!). Interleaving and de-interleaving to make distribution of raw channel errors amenable to FEC.
- Control functions: Cellular systems have a huge amount of event-driven signal processing and adaptation of that signal processing (Call setup/teardown, closed loop control of transmission time/power/frequency/code and channel equalisation in dynamic conditions for example)

These applications typically consist of a mixture of DSP functionality which is both stream based and block based. In addition there is a requirement for distributed control of the DSP operations. The picoArray™ architecture provides stream based support by having a large number of processors which have small amounts of local instructions and data. Block based support is provided by having a number of the processors which have increased local data and if this is insufficient then there is also access to an external memory. Conventional architectures (and some newer ones) force designers to fit the algorithm to the architecture rather than choose the algorithm implementation to suit the performance required (e.g., block-based approaches incur latency overheads unpalatable in some systems but conventional DSPs are only efficient when operating block based).

Previous approaches to these applications have used ASIC, DSP and FPGA to produce a solution. Often they require a combination of these technologies which in itself produces problems with co-design and co-simulation. In addition, the most cost effective method, ASIC, is extremely inflexible, which causes problems when wireless standards change or algorithmic improvements need to be made. The DSP/FPGA solution can be flexible but tends to be expensive, power hungry and combining the two technologies appears never to be simple. The picoArray™ produces a reprogrammable solution to a wide range of these applications using a single tool chain, rather than having to combine DSP tool chains with FPGA and perhaps ASIC.

To provide some idea of the complexity of this type of application figure 6 shows the major software components in a typical UMTS baseband system. Many of the blocks in the diagram are complex systems in their own right and the blocks may need to be replicated for each user of a base-station.

4 Programming the picoArray™ - the tool chain

The design of the tool chain strongly reflects the application domain of the picoArray™ and this has been used to simplify the tool flow. For example, the C compiler deals with the code for a single processor and no attempt is currently made to automatically parallelise code across multiple processors. In addition there is considerable use of assembly language coding and experience has shown that this can be achieved efficiently (from the user's perspective) due to the high degree of replication of the code.

The picoArray™ is programmed using a mixture of VHDL [5], ANSI/ISO C and assembly language. The VHDL is used to describe the structure of the overall system, including the relationship between processes, and the signals which connect them together. Each individual process is programmed in conventional C (albeit with additional communication functions), or in assembly language. A simple example is given in figure 7.

A comprehensive tool chain exists to convert the input VHDL into a form suitable for execution on one or more picoArray™ chips. The tool chain includes a compiler, an assembler, a VHDL parser, a cycle-accurate simulator, debuggers, a place-and-switch tool, design partition tools and verification tools. The relationship between these tools is shown in figure 8. The following sections examine each of these tools in turn.

4.1 VhdlParser

The VHDL parser is the main entry point for the user's source code. A complete VHDL design is given to the parser, which coordinates the compilation and assembly of the code for each of the individual processes. An internal representation of the machine code for each processor and its signals is created.

4.2 C Compiler

The C compiler is a port of the GNU Compiler Collection (GCC) [6]. A few simple extensions have been provided to support communication, but the compiler otherwise supports conventional ANSI/ISO C. GCC is designed primarily for 32-bit general purpose processors capable of using large amounts of memory, making it a challenge to support 16-bit embedded processors with just a few kilobytes of memory. The compiler uses a Deterministic Finite Automata scheduling algorithm [7] to generate efficient VLIW schedules.

In addition to being called by the VhdlParser to compile code for processes, the compiler may be used stand-alone to generate library files. These libraries can be linked with the processes in a VHDL design, allowing effective code reuse.

4.3 Cycle accurate simulation

The cycle accurate simulator can be constructed directly from the output of the VhdlParser, since there is no need to determine how a design must be partitioned between chips, or how processes are allocated to processors. The simulator can be used in two modes, functional or cycle accurate, depending on the requirements of the user.

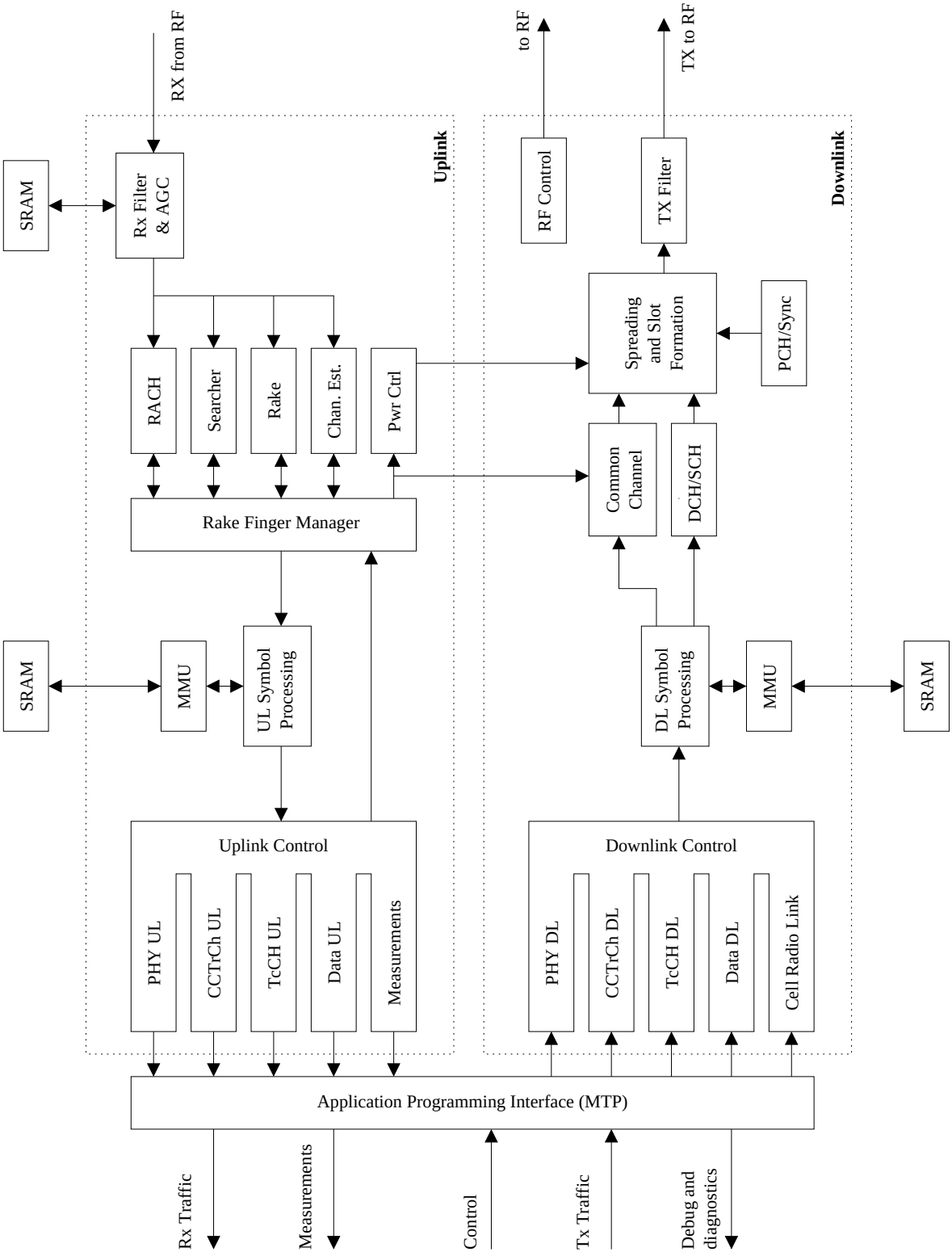

Figure 6: Diagram showing software components in a typical UMTS baseband system.

```
------------------------------------------------------------------------
entity Producer is                        -- Declare a producer
  port (channel:out integer32@8);         -- 32-bit output signal
end entity Producer;                      --   with @8 rate

architecture ASM of Producer is           -- Define the 'Producer' in ASM
begin MEM                                 -- use a 'MEM' processor type
  CODE                                    -- Start code block
    COPY.0 0,R0 \ COPY.1 1,R1             -- Note use of  VLIW
  loopStart:
    PUT R[0,1],channel \ ADD.0 R0,1,R0    -- Note communication
    BRA loopStart
  ENDCODE;
end;                                      -- End Producer definition.

------------------------------------------------------------------------

entity Consumer is                        -- Declare a consumer
  port (channel:in integer32@8);          -- 32-bit input signal
end;

architecture C of Consumer is             -- Define the 'Consumer' in C
begin STAN                                -- Use a 'STAN' processor
  CODE
  long array[10];                         -- Normal C code

  int main() {                            -- 'main' function - provides
    int i = 0;                            --    entry point

    while (1) {
      array[i] = getchannel();            -- Note use of communication.
      i = (i + 1) % 10;
    }

    return 0;
  }
  ENDCODE;
end Consumer;                             -- End Consumer definition

------------------------------------------------------------------------

use work.all;                             -- Use previous declarations

entity Example is                         -- Declare overall system
end;

architecture STRUCTURAL of Example is     -- Structural definition
  signal valueChannel: integer32@8;       -- One 32-bit signal...
begin
    producerObject: entity Producer       -- ...connects Producer
      port map (channel=>valueChannel);
    consumerObject: entity Consumer       -- ...to Consumer
      port map (channel=>valueChannel);
end;
------------------------------------------------------------------------
```

Figure 7: Example source program

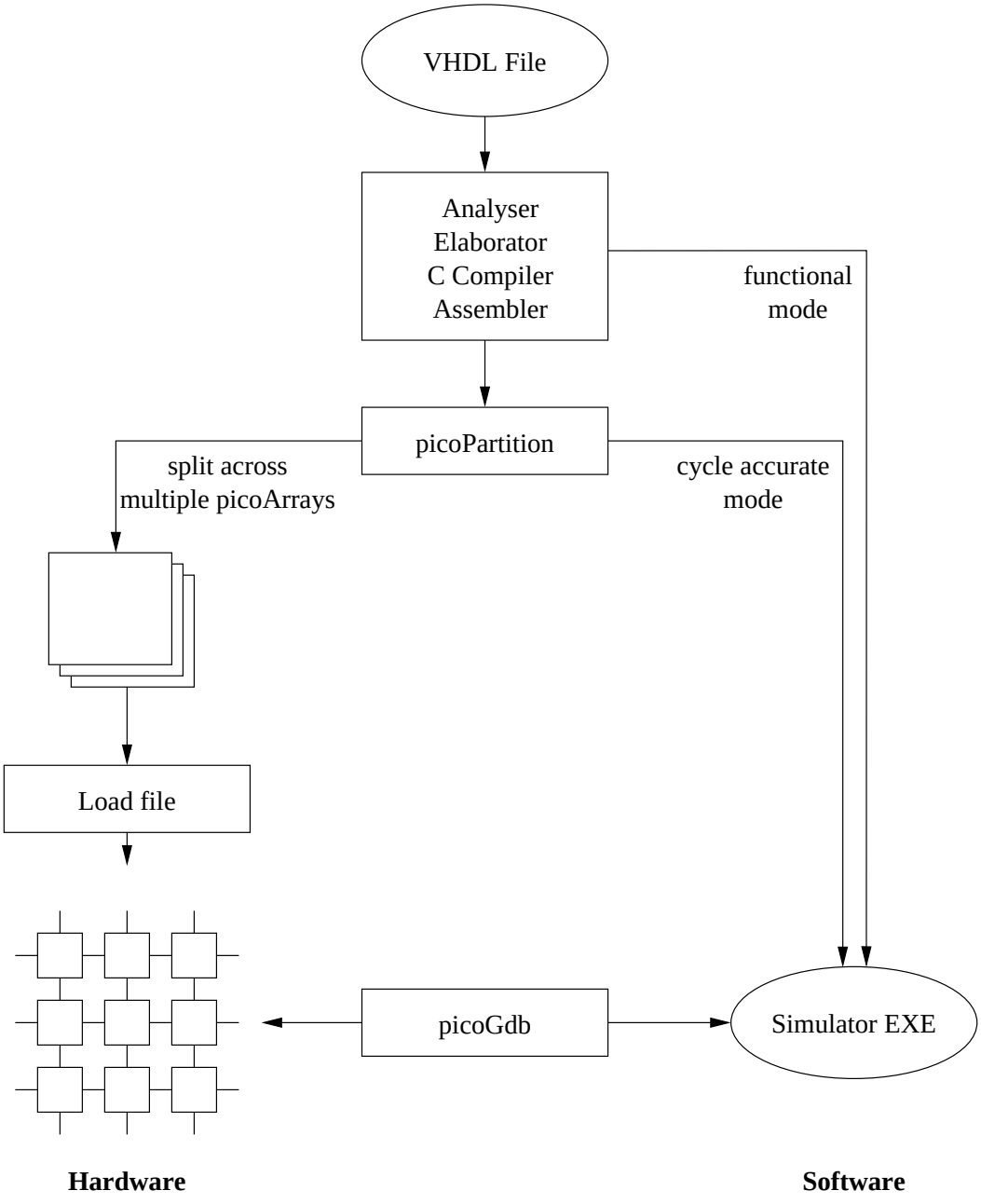

Figure 8: Tool Structure

In functional mode it can model the logical functionality of a design with cycle accuracy only provided within each individual processor, the timing of the communications is not modelled accurately. This can be used early in the design process as it is not necessary to determine where processes are to be placed on the hardware and even which chip they will reside on.

In cycle accurate mode the design must be partitioned between chips (see Section 4.4), processes must be placed onto specific processors and the signals must be routed on the picoBus™ (see Section 4.5). This allows inter-chip timing information and signal timing information to be back annotated into the simulation to produce a fully cycle accurate simulation.

4.4 Chip partitioning

If a design requires more processors than are available in a single picoArray™, the design must be partitioned across multiple chips. This process is currently manual, with the user specifying which processes map to which chip, although the splitting of signals between the chips is automated.

4.5 Place and Switch

Once a design has been partitioned between chips a process akin to place and route in ASIC designs has to be done for each chip. This assigns a specific processor to each entity in the design and routes all of the signals which link entities together. The routing must use the given bandwidth requirements of signals while routing. The routing algorithm should also address the power requirements of a design, by reducing the number of bus segments that signals have to traverse, enabling unused bus segments to be switched off. This process is performed using the Plastic (PLace And Switch To IC) tool.

When a successful place and switch has been achieved a “load file” can be produced which can be loaded directly onto the hardware.

4.6 Debugging

The debugging tools allow an entire design to be easily debugged, either as a simulation, or using real hardware. The tools support common debugging operations such as setting breakpoints, single and multi-step execution, halt-on-error, status display, and memory/register tracing. A port of the GNU Debugger (GDB) [8] allows source-level debugging, using either C or assembly language. For flexibility, both graphical and command-line interfaces are provided.

Debugging parallel processes is notoriously difficult, so some special debugging operations are provided to aid the user, including “spies” and traffic analysis. Spies are special processes which are inserted in the middle of a signal, and allow the flow of data through that signal to be monitored. Traffic analysis is used to identify how much bandwidth is being used in different parts of the picoArray™, and can find hot-spots in the bandwidth usage, or detect deadlock.

5 Current Status and Performance

The current implementation of the picoArray™, the PC101, runs at 160MHz and is implemented using 130nm technology. A PC101 device running at 160 MHz can execute 206

GIPS, excluding acceleration from the application-specific instructions. A demonstrator system consisting of 16 devices is capable of approximately 3,300 GIPS. Comparison of performance with other architectures is notoriously difficult but a Texas Instruments TMS320-C6416 running at 600 MHz can execute approximately 9.6 GIPS.

The main application area that has been chosen to demonstrate the technology is a UMTS base-station and much of this has been coded using PC101 hardware and the tool chain described above. First silicon for PC101 was received in December 2002 and the first telephone call through a demonstration base-station using this silicon was made in May 2003. This uses a system consisting of 4 PC101's with 740 processors being used.

6 Conclusion

The basic ideas behind the picoArray™ concept together with the links to the application domain have been described. A pragmatic approach has been demonstrated to the tool chain in which the expertise of the user and the specific application domain has been used where possible rather than trying to create "magic" tools which would be complex, time consuming to create and may never materialise. This is not to say that approaches to automatic partitioning and automatic code scheduling are not being investigated but picoArray™ technology does not stand or fall on the basis of the existence of such tools.

The creation of a major application, a UMTS base-station, in only a few months using the technology demonstrates the power of the approach taken in terms of ease of use due to the single design flow.

7 Acknowledgements

Thanks in particular to Peter Claydon and Doug Pulley who co-founded picoChip Designs Ltd. Thanks to all who have worked on the huge variety of tasks that have made PC101 systems a reality.

References

[1] P. Claydon. Picoarray Switch Matrix. Patent number GB 0030993.0, 2002.

[2] 3GPP. *3GPP TS25 Series (FDD + TD-SCDMA)*.

[3] IEEE. *802.16 IEEE Standard for Local and metropolitan area networks*.

[4] TIA/EIA. *TIA/EIA-IS-2000 series*.

[5] Peter Ashenden. The Designer's Guide to VHDL. Morgan Kaufmann, ISBN 1-55860-270-4, 1996.

[6] Richard Stallman. Using and porting the GNU compiler collection. ISBN 059510035X, `http://gcc.gnu.org/onlinedocs/gcc/`, 2000.

[7] Vladimir Makarov. The finite state automaton based pipeline hazard recognizer and instruction scheduler in GCC. The 2003 GCC Developers' Summit Conference Proceedings, `http://www.linux.org.uk/~ajh/gcc/gccsummit-2003-proceedings.pdf`, May 2003.

[8] Richard Stallman, Roland Pesch, and Stan Shebs. Debugging with GDB. ISBN 1882114884, `http://sources.redhat.com/gdb/current/onlinedocs/gdb.html`, 2002.