Enable++: A second generation FPGA processor[*]

H. Högl, A. Kugel, J. Ludvig, R. Männer, K.H. Noffz, R. Zoz
Lehrstuhl für Informatik V
Universität Mannheim

Abstract

Two years of experience with the two prototype FPGA processors Enable-1 and DecPeRLe-1 reveal that field programmable processors are the best choice for realizing a data-driven second level (L2) trigger for ATLAS. This paper presents Enable++, a 2nd generation FPGA processor that offers several substantial enhancements to the previous systems:

In order to meet the varying demands of all ATLAS subdetectors Enable++ is structured into three different state-of-the-art modules for providing *computing power*, flexible and high-speed *I/O communication* and powerful *intermodule communication* with a raw bandwidth of 3.2 GByte/s by an active backplane. The technical realization of all three modules is guided by the maximum usage of field programmable logic. The actual demand of computing- and I/O-power can be satisified by the number of modules plugged into the crate.

Enhanced features of Enable++ comprise the configurable processor topology provided by programmable crossbar switches. In combination with the 4×4 FPGA array and 12 MByte distributed RAM the Enable++ computing core offers strongly increased and scalable computing power. For building new applications the system provides a comfortable programming and debugging environment consisting of a compiler for the C-like hardware description language spC, a simulator and a source level debugger for hardware design. The goal in planning the hardware design environment for Enable++ from scratch is to transfer established methodologies in software design to the design of digital logic.

The most computing intensive tasks in L2 triggering are the feature extraction algorithms. From experience with Enable-1 we expect that Enable++ surpasses modern RISC processors by a factor of 100 to 1000. The new processor will be available in summer 1995 and the participation in the common ATLAS run in Sept. 1995 is foreseen.

[*]This work has been performed within the RD-11 and ATLAS collaborations and has been supported by the Bundesministerium für Forschung und Technologie (BMFT) under grant PH/-RS6-93/09 and the Gesellschaft für Schwerionenforschung (GSI) in Darmstadt, Germany, under grant MAMÄK.

1 Introduction

The concept of a data-driven 2nd level trigger in ATLAS requires processors keeping up with the high event rate of 100kHz. These processors have to be fully programmable in order to allow algorithm optimizations even after the experiment has started to run. The flexibility of this type of data-driven processor has to be high enough to serve as a L2 trigger for all subdetectors of ATLAS.

Reconfigurable hardware built with field programmable components is an ideal concept for constructing data-driven trigger processors combining both the speed of a hardware and the flexibility of a software solution. This type of processor consists of a computing core using a set of Field Programmable Gate Arrays (FPGAs) which are configurable by software to form a hardware implementation of an algorithm. Two exemplary implementations of this concept are Enable-1[1] and DecPeRLe-1[2].

In 1991 the EAST RD/11 collaboration carried out a benchmark competition in order to find candidates for a L2 trigger processor. The task was to implement the TRT algorithm within given requirements. The only system that met the full benchmark requirements was Enable-1 [BBB+93]. During the last two years the FPGA processors Enable-1 and DecPeRLe-1 were studied intensively in beam tests and simulations [BBB+94]. Enable-1 has demonstrated its ability as a feature extractor for a full-scale TRT running under LHC conditions. On DecPeRLe-1 several algorithms for test detectors (TRT, SCT, Calorimeter) were successfully implemented.

This paper is the result of two years of in-depth experience with FPGA processors for EAST. It presents Enable++ as the genuine 2nd generation of FPGA processors for L2 triggers. The new processor combines advantages from Enable-1 and DecPeRLe-1 and introduces several new features. Section 2 gives a brief overview of the hardware architecture of Enable++. The most important improvement is that Enable++ is programmable in a high-level language similar to C, that makes application devolopment more convinient. This programming environment is discussed in section 3. Section 4 deals with the expected performance of Enable++ as a data-driven trigger. The last chapter closes with an overview over the current status of the project.

2 The Enable++-System

The general view of the whole system will be a standard workstation which is connected to the Enable++ hardware. All hardware parts are contained in one (or up to three) 9HU sized VME crates. The workstation will remain in any respect an ordinary 'working environment', enhanced with additional

[1]Built at Lehrstuhl für Informatik V, Universität Mannheim [NZK+93].

[2]DecPeRLe-1 was developed by the Paris Research Laboratory (PRL), a research laboratory of Digital Equipment (DEC) [BRV93].

capabilities by the fact that it is connected to a powerful FPGA processor. For this purpose one can use either a standalone model or a VMEbus based plug-in board. The physical connection depends on the type of frontend: If a VMEbus type workstation is used, one has the advantage of connecting both by a high bandwidth system bus (e.g. SBus) *and/or* by a serial bus which is part of the systems *service network* and is realized by a transputer link. Using a standalone workstation one is restricted to use only the serial bus because of the distance between both systems. Both buses offer the same functionality, which means that both have the ability to access all system resources.

After connecting the frontend workstation to Enable++, the user will concentrate on writing applications. For this purpose he will deal with the Enable++ Development Environment (EDE) which offers all the functionality for breaking down a high-level specification of a new application to an Enable++ configuration. The user can choose among different functions as there are *compiling* of a system description, hardware *source level debugging* and *simulation*. EDE offers with spC its own brand of hardware description language. It is derived from both C and Hardware-C and flavoured with additional constructs for describing systolic data processing. spC is the attempt to realize a hardware description language which is intermediary with respect to abstraction level and hardware control. Because EDE is planned to fit tightly in a UNIX environment, the user will have in addition dozens of useful and well established tools for free.

After outlining the main 'look-and-feel' of the whole Enable++ system, this section will proceed with a closer look at the hardware components of Enable++. As depicted in figure 1 there are three different types of components:

- Computing Array Boards (CA)
- I/O Boards (I/O)
- an active Backplane (AB)

Computing power is mainly located on the Computing Array Boards. On each board a matrix of FPGAs is connected with high-speed RAM and a network of I-Cube[3] crossbar-switches (FPIDs) building an extremely high-powered computing core which can be adapted to a broad range of different topologies.

The I/O Boards deal with incoming and outgoing data from different sources and sinks (SBus[4], FibreChannel[5], HIPPI[6], SCI[7], etc.) and may use different types of receivers and transmitters. A single board uses four different ports for

[3]Field Programmable Interconnect Devices (FPIDs). We use the IQ240, which offers 240 connectable I/O pins.

[4]SUN System I/O Expansion Bus

[5]FibreChannel is a high-speed optical connection, 1062 MBd

[6]High Performance Parallel Interface with 100 MByte/s data rate.

[7]Scalabel Coherent Interface

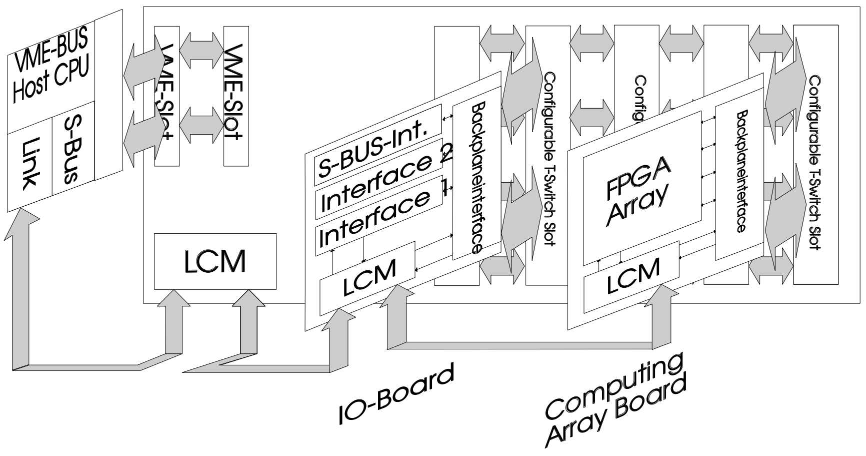

Figure 1: Enable++ system overview

data transmitters or receivers. By the use of FPIDs another convenient usage of this board is that of a router. Source and destination of all four ports can be other ports or the active backplane.

The concept of an active backplane was realized after investigating generalized backplane concepts. The backplane now constitutes a field programmable module in the same way as the other two components. It connects I/O boards and CA boards and is mainly responsible for high-speed data transfer between them. Its open interconnection scheme has the ability to implement a variety of communication models in multiprocessing applications. A possible protocol could be token-ring based with a raw bandwidth of 3.2 GByte/s.

Furthermore both CA and I/O boards are supplied with a novel bus interface architecture based on FPGAs and local accessible dual-ported RAM. This interface is capable of doing high-speed pre- and postprocessing of data flowing to and from the board (e.g. formatting) and of synchronizing the data flow between the on-board logic and the backplane.

For the purpose of system maintenance we introduced the concept of Local Control Modules (LCMs) which are connected via an autonomous service network. All of the modules introduced above contain a LCM module which serves for standalone board level test, board initialization, board monitoring and debugging support.

2.1 Computing Array Boards

The main functional parts of the Computing Array Board (CA) are the FPGA array and the bus interface. In terms of absolute FPGA resources each CAB has a complexity of approx. 288,000 gates (using the Xilinx figures for sixteen

XC4013- and eight XC4010-type FPGAs). The FPGA array consists of 4×4 XC4013-type FPGAs which are connected as shown in the left part of figure 2 (for the sake of clarity we omitted the connections to the lower half of the bus interface).

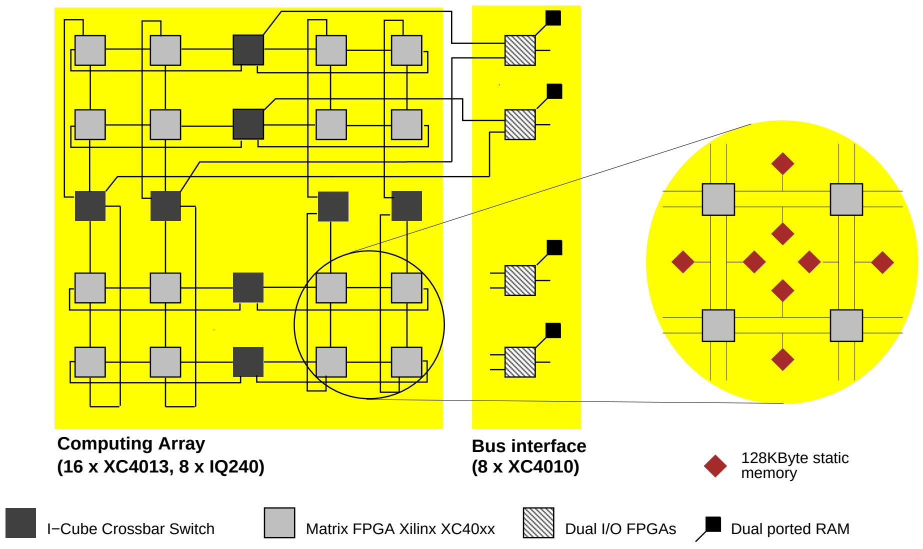

Figure 2: Major building blocks of a Computing Array Board.

Besides nearest neighbour connections, FPGAs are connected through I-Cube crossbars so that different global topologies can be achieved via configuration. Possible topologies are for example linear arrays, rings, cubes and hypercubes (see figure 3). This highly regular connection scheme allows for very fast and broad pipelined flows of data as needed with systolic algorithms. We believe that with this architecture high utilization is achievable for many regular systolic designs and that even random logic can be mapped on this topology with acceptable losses. In comparison with Enable-1 we have nearly the same count of configurable logic blocks (CLBs) due to less but higher integrated FPGAs. By the new design we end up with improved routing possibilities, higher I/O bandwidth and extended interconnect possibilities.

In addition there will be 96 high-speed 128 kByte synchronous SRAM chips on board. This is in total 12 MByte of extremely fast local RAM. These RAMs are located at the direct interconnections between neighbouring FPGAs as shown in the right part of figure 2. A scheme like that is suitable for pipelined lookup tables (FPGA1-RAM-FPGA2) with up to 17 input bits and 16 output bits and general read/write RAM. In both cases these RAMs have no wait state and one clock cycle latency at 50 MHz system clock. The total throughput is thus 4.8 GBytes/s.

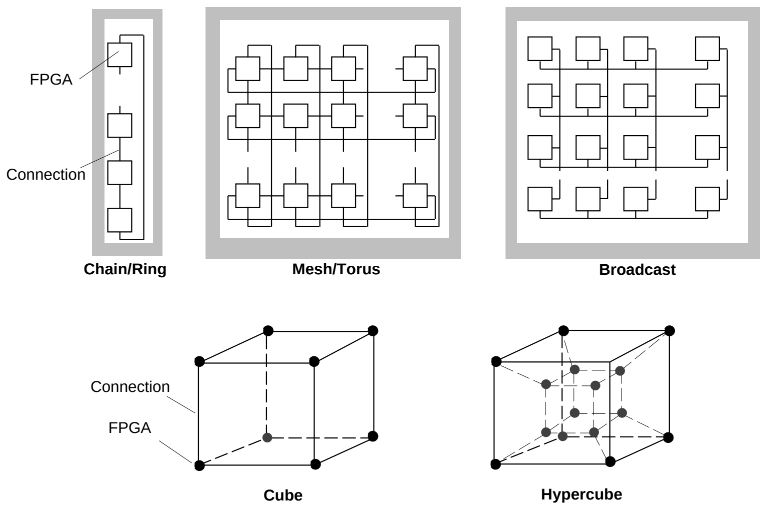

Figure 3: Examples of possible configurations.

The computing array is connected via the crossbars to a bus interface (shown in the middle part of figure 2), comprising eight XC4010-type FPGAs and fast dual-ported RAMs. This circuit decouples the synchronously operating array from asynchronous bus protocols on the backplane and buffers incoming or outgoing data. Connection to the backplane is done by eight 40-bit buses (32 bit plus 8 bit control) which can be operated synchronously up to the full clock speed of 50 MHz. This equals a data rate of 1.6GByte/s.

2.2 Enable++ I/O subsystems

The Enable++ I/O subsystems connects the computing core of Enable++ to its real-world environment via high-speed datapaths. It is scalable to achieve an overall data throughput capability equivalent to the installed computing power. To provide a maximum flexibility in connecting to external data sources or destinations a modular design of the I/O boards was selected. Every I/O motherboard can carry up to four I/O daughterboards, each one implementing a specific transmission protocol and media access. The LCM module described in section 2.4 can be used as a fifth virtual I/O port in addition to its primary functions. Together with the four buffered ports connecting to the backplane bus, there is a total of 9 data channels on one I/O motherboard. Every channel can be arbitrarily configured as input or output. Programmable logic (FPGAs and FPIDs) serves for routing and preprocessing of the data. All 9 data channels can deliberately be connected to any one or more of the other channels.

With these features the I/O system is well suited to perform complex routing operations which go far beyond a trivial point-to-point connection. Possible operations are interleaving of two external channels to one of the backplane's subbuses, replication of incoming or outgoing data, even format or protocol conversion can be done. The on-board bandwidth for every one of the 9 channels is 50 MHz by 32 bits (200 MByte/s). Although the 4 backplane ports can keep up with this speed, this value is obviously limited in the case of the outbound channels by the properties of the specific I/O daughterboard. I/O daughterboards to be manufactured in the nearest future will connect to the HIPPI channel and to the SBus of SUN-type workstations. There are also plans for ATM[8] and FibreChannel interfaces. For special purposes there will also be a module providing several Megabytes (approx. 32 . . . 128) of buffer memory with a bandwidth of 200 Mbyte/s in block access mode.

2.3 Backplane

Handling as high data rates as done by Enable++ is viable only with a special kind of bus system. Since Enable++ is intended to be a multiprocessing system care must be taken to avoid any kind of bottleneck. For this reason a configurable active backplane bus is integral part of the system. It will have a multiple-T topology where at every slot the same kind of router circuit will buffer, split, bridge and route signals coming from neighbouring slots. Thus static and dynamic reconfiguration of the system is possible. It furthermore enhances overall data rates since an active routing scheme makes it feasible to use wider buses on the backplane than on the Computing Array Boards and have other than nearest-neighbour connections. Every slot is splitted horizontally into eight 40 bit wide subbuses, each having 32 bit of data and 8 bits for control. With this architecture it is straightforward to implement systolic datapaths with the full system speed of 8×200 MByte/s data-rate in total.

2.4 Enable++ Local Control

One important feature of the Enable++ system is the implementation of a uniform local control module (LCM) on each of its module boards. The main elements of this LCM are a T425 transputer with 4 to 32 MByte dynamic RAM and 4 transputer links plus an XC4005-type FPGA. Mechanically the LCM is a piggy-back daughterboard to be plugged onto the specific Enable++ motherboard: Computing Array, Backplane or I/O board respectively. An additional LCM may be used in conjunction with a special interface board in the host computer, if needed. Each LCM can be connected via its transputer links to up to 4 other LCM modules or other link interfaces. Typically all LCM modules in the system form a daisy-chain by connecting via two links each to its left and right neighbour. At least one link connects from the daisy-chain to the

[8]ATM is a network protocol used in high-speed serial connections (155 MBd, 622 MBd available)

host computer. Through the FPGA the transputer gains access to all hardware resources of the motherboard it is controlling. Additionally the FPGA has the capability to interrupt the transputer on an internal event or on request from the motherboard, and to perform direct-memory-access to the dynamic RAM.

The tasks of the LCM are

- standalone board level test
- board initialization
- board monitoring
- debugging support

2.4.1 Standalone board level test

In this basic mode the LCM is connected to a host computer via one of its links. The transputer performs JTAG/boundary-scan compliant board level testing plus special test algorithms, if necessary. Our goal is to achieve 100% test of all connections and functions of the motherboard.

2.4.2 System initialization

The host computer transfers all configuration data to the LCM module which first stores it in its internal memory. For a full-scale Enable++ CA board this will take about 1.5 Mbyte of memory. Depending on the memory resources on the LCM, alternate configuration data sets may be transmitted subsequently to provide means for dynamic configuration changes. The LCM transputer performs configuration of the motherboard by selecting any group of up to 32 configurable logic chips and transferring the data either under program control or via direct-memory-access. Typical configuration or reconfiguration times will be less than 50 ms for a full-scale Enable++ CA board. With a large memory to store different configuration data sets the LCM may perform complete or partial dynamic reconfigurations of the motherboard, either on demand by the host computer or after an interrupt generated by the motherboard. Algorithms may take advantage of this self-modifying-hardware capability.

2.4.3 Board monitoring

The FPGA chip on the LCM can be connected to some user defined status lines on the motherboard. Application specific designs can be developed for the LCM to trigger on certain conditions on these status lines. Various timers, counters or logical equation checks may be implemented. As the transputer is not fast enough to keep up with the speed of the motherboard, the FPGA brings the real-time capability to the LCM module.

Any kind of error handling and logging procedures can be run on the transputer. Samples of data can be taken for online monitoring, either small data sets at full speed or arbitrary data sets with a reduced speed. Thus data can be transferred to the host computer via the transputer link without interfering with the main datapath on the backplane bus.

2.4.4 Debugging support

Support for debugging relies on several properties of the components used on all boards or modules and on special features of the LCM:

- The clock system of each motherboard can be controlled by the LCM
- All FPGA and FPID components support full JTAG boundary scan capability
- All FPGA components support a non-destructive readback capability of all internal states and memory locations/registers
- Each motherboard provides a set of programmable status lines
- Each LCM runs a special debug task communicating with the Enable++ system debugger
- High-speed trigger conditions on the motherboard can be handled by the LCM
- All LCM modules form a link-based network independent from the main datapath.

The LCM performs the mapping of source level node names to local hardware resources with the help of the debugging database (see chapter 3). The average access time from the LCM to any given resource on its motherboard varies with the location of that resource (memory, I/O buffer, FPGA internal register) but will not exceed a few milliseconds - fast enough for interactive debugging. As the data flow between the host computer and the LCM modules consists mainly in control information the speed of 20 MBd on the link network will be fairly sufficcent.

Debugging of the Enable++ system can be performed not only board-by-board but on system level and with the comfort of a high-level debugging tool. The LCM assists the main debugging task running on the host computer by virtualizing the hardware on the Enable++ board components.

3 The Design Environment

One major criterion in the design of the Enable++ machine is the underlying general hardware architecture which allows a broad range of applications in different fields such as pattern recognition, number crunching, fast networking and logic simulation. Because the system's configuration space is enormous, it is most important to supply the Enable++ hardware environment with powerful tools for designing and debugging new applications. Our goal is to establish a toolset which shifts the design process for hardware as far as possible to a standard which has been well established for years in software design.

The EDE (Enable++ Development Environment) we are currently building is designed with the following principles in mind:

- The procedure of building an application for Enable++ should be very similar to building an ordinary software application.
- The operation of Enable++ is tightly bound into a program running on the host workstation. Both host and Enable++ functionality is specified in one common set of source files.
- A source level debugger allows comfortable testing and debugging of Enable++ applications *while* the hardware is actually operating.
- A simulation tool provides the possibility to test Enable++ algorithms completely prior to any hardware implementation.

3.1 Overall structure of the design environment

The usefulness of a large FPGA processor like Enable++ depends heavily on how easily the system can be used and adapted to new applications. In an extreme scenario a human operator would solely write ordinary software without being aware of using application specific reconfigurable logic. An intelligent design system would decompose the software specification and automatically map parts of the software to hardware. With Enable++ we will provide a design environment which offers an intermediate solution to the design problem which is much more useful in application design than the above 'radical' solution: Both hardware and software is explicitely specified in one common set of source files. In case of the software part on the host computer the language is C, in case of the hardware part it is spC (systolic parallel C, a derivative of both C and Hardware-C), which is designed for describing systolic and parallel logic systems.

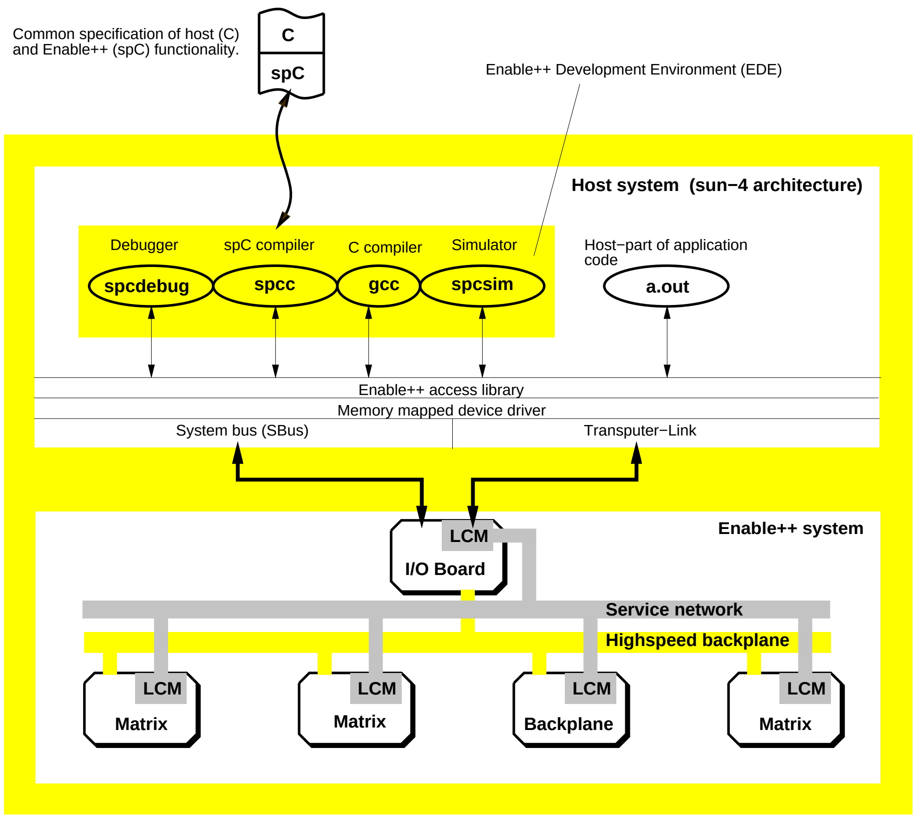

Figure 4: Overview of Enable++ Development Environment EDE.

As shown in figure 4, the EDE consists mainly of four different tools. The spC compiler `spcc`, the host C compiler `gcc`, the hardware debugger `spcdebug` and the hardware simulator `spcsim`. The principal processing of the two paths (host software/Enable++ hardware) of an application program is done by the compilers `gcc` and `spcc`.

The paradigm shift of specifying hardware by software makes well established rules in traditional software design applicable in hardware design. One of the most important rules states that a programming system is only as useful as its debugging facilities. We adopted this rule and plan the source level debugger `spcdebug` to have complete control of the hardware design in the debugging phase similar to the debugging capabilities of C source level debuggers (`gdb`, `ups`) in traditional software design. For this purpose `spcdebug` makes use of the LCM service network to access Enable++ hardware.

With the simulator `spcsim` we want to enhance the development environment with a tool to verify the behavior of algorithms prior to any mapping in Enable++ hardware.

Figure 4 also shows the approach we chose in communicating between host and FPGA processor. There are in principle two possibilities for connecting a host workstation to the hardware. First, if the host computer is a standalone workstation, then the connection is established via a Transputer Link to the Service Network. Second, a VMEbus based host machine can be plugged into the Enable++ crate with either Link or SBus connection or both to an I/O board. A layered communication software provides an abstract and structured way of accessing and exchanging data between both entities. The access point for services like EDE and application programs is the Enable++ access library. The low-level device driver and physical I/O are in this way hidden from the application programmer.

3.2 The development cycle

As shown in figures 4 and 5, a mixed description consisting of C and spC code is the starting point for a new design. A preprocessor handles the separation in C and spC code.

C path: The C code output by the preprocessor constitutes the host computer part of the application program. For accessing the FPGA processor the code has furthermode to be linked with the Enable++ access library.

spC path: The spC code is processed with the `spcc` compiler which uses a configuration file for controlling various task specific hardware parameters (e.g. configuration of the FPIDs). `spcc` outputs code for three different purposes:

a) <u>VHDL code</u> for specifying the functionality of the FPGAs.

   `spcc` converts the spC description to VHDL code. We chose this format because of the recent availability of commercial CAD tool which map VHDL code to different FPGA technologies, e.g., Synopsys and Asyl+. With one of these tools the VHDL code is further processed to Xilinx configurations (LCA files).

b) A <u>simulator database</u> which can be "executed" with the simulator kernel `spcsim`. The simulator database contains data structures which are built during compilation with `spcc`.

c) A <u>debugger database</u> which allows source level debugging of the Enable++ hardware with `spcdebug` (see also section 2.4.4). For source level debugging it is essential, that the elements of the synthesized logic are mapped to corresponding code fragments of the initial spC description.

Relevant parts of the debug database are distributed across the LCM network. The debugger task on each LCM module transparently performs the

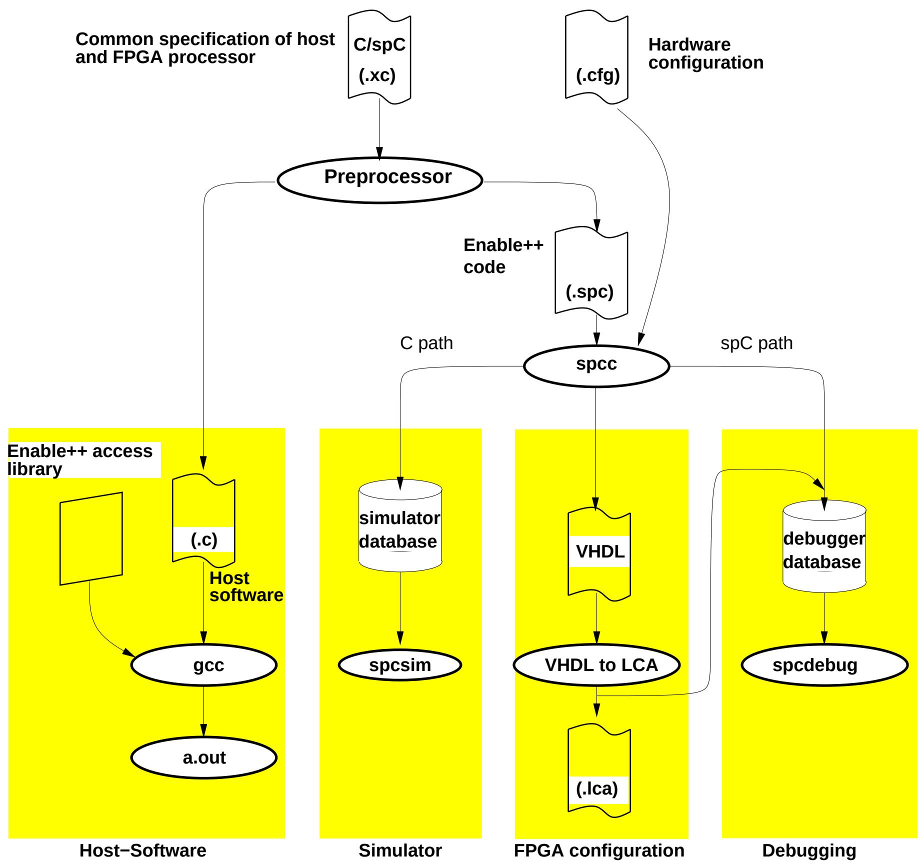

Figure 5: Design paths for building Enable++ applications.

accesses to the hardware ressources, setting and checking of breakpoints and inspecting/modifying of variable registers.

4 Performance

After description of hardware and software we will now discuss the performance of Enable++ as a part of the L2 trigger. This discussion is based on a set of L2 benchmark algorithms defined in [BHL94]. All benchmark algorithms are defined as C programs representing the actual physics and detector assumptions and are believed to be representative for algorithms being implemented in ATLAS. The strict specification of all algorithms details including input data formats allows a performance comparison of different architectures.

As long as the Enable++ hard- and software is not completed and direct measurements therefore impossible the benchmark algorithms are formulated in spC, and speed as well as resources are manually estimated. This method is simplified by the pipelined structure and the lack of data dependencies in the algorithms. The accuracy of the performance estimation is acceptable (between 10% and 20%) as experiences with a similar method for Enable-1 have shown.

In a pure data-driven L2-trigger as well as in a hybrid system (consisting in a data-driven feature extractor combined with a farm based global decision) the feature extraction is the most computing intensive and therefore time critical operation. For that reason we will focus in this chapter on the feature extraction algorithms only: the calorimeter-, the drift time TRT-, and the Silicon Tracker (SCT) algorithm. The principle idea of implementation combined with the required FPGA resources is given for each feature extraction algorithm, table 1 at the end of this section summarizes the performance estimation on Enable++. For a detailed description of the algorithms refer to [Leg94] (calorimeter, TRT) and [KM94] (SCT).

4.1 Transition Radiation Tracker (TRT)

The trigger task of the endcap TRT is to determine the most significant electron track in the ϕ/z projection of the detector. Particle tracks appear as straight lines of slope dϕ/dz in the ROI image; the ROI dimensions are 96 planes (of constant z) times 16 straws (along ϕ). Electrons are identified by their transition radiation production rate along a track (corresponding to the ratio of high and low threshold hits in the ROI).

The basic idea of the TRT algorithm consists in a Hough transform for track recognition and an weighted maximum finding for the electron identification. Input data arrive as a zero-suppressed list of 16 bit pixel coordinates. In the implementation of the Hough transform this pixel list is asserted to a huge lookup table of 64k$\times$ 256 bins where each bin corresponds to a track of a certain slope and starting angle. After histogramming all bins for the complete list (independently for both high and low threshold), a weighted maximum of all histogram channels is evaluated.

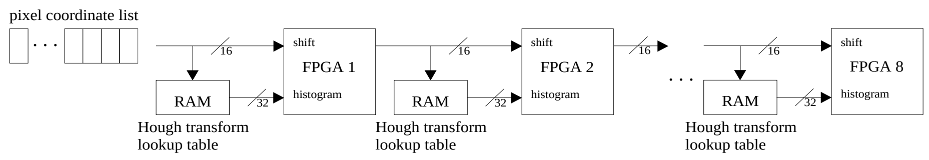

Figure 6: Implementation of the TRT algorithm.

To implement the TRT algorithm on Enable++ the coordinate list is fed into a pipeline of FPGAs and accompanied RAMs as shown in figure 6. Each stage of the pipline contains a part of the full Hough transform lookup table (implemented in the RAM) and the corresponding histograms channels (implemented in the FPGA). Note that the time to build the Hough space is equal to the time required for the I/O of the coordinate list (exept a minor latency of 0.2 μs).

The whole algorithm (Hough space histogram building as well as evaluation) can be performed within $9 \mu s$. In term of logical resources per FPGA it consumes 55% for the 64 9 bit histogram counters, 2% for the evaluation pipeline and about 10% for control and datapath logic, leaving 1/3 of the FPGA unused. Because the algorithm only uses one half of the FPGAs and RAMs, two TRT ROIs can be executed concurrently on a single Enable++ board. The performance estimation for the TRT is very accurate, because the needed circuitry is very similiar to that implemented on Enable-1.

4.2 Silicon Tracker (SCT)

Similar to the TRT algorithm a Hough transform is used to find tracks in the r-ϕ-projection of the SCT (ROI size is 4 layers of 1000 bins each). Because both TRT and SCT use a zero suppressed pixel coordinate list as input, the principle implementation shown in figure 6 is used for the SCT too. The major differences to the TRT are the 8 times wider (16 times less deep) Hough space (4k$\times$ 2048 bins) and a different histogram evaluation algorithm .

To save RAM resources four different Hough lookup tables are implemented into one RAM and addressed sequentially for each list entry. This leads to a speed reduction by a factor of four that is acceptable because of the low occupancy of the SCT (1% ...2.5% corresponding to 40 ...100 list entries). As for the TRT algorithm the execution time is strongly dominated by the time to generate the histograms. Table 1 lists the execution time for the SCT in different occupancies assuming a board frequency of 50 MHz. The logical resources used per FPGA consists in 128 3 bit counters for the Hough histograms (consuming 40% of the logic), correlator logic for the histogram evaluation (7% FPGA logic), and some datapath and control logic (below 15% logic).

4.3 Calorimeter algorithm

The calorimeter input data consists in 3 electromagnetic (em) and 4 hadronic(ha) layers of size 20$\times$20 (em) and 5$\times$5 (ha). The features that have to be extracted are the center of gravity of the total energy deposition, the hadronic energy fraction and the 2nd moment for every electromagnatic layer. In contrast to the previous algorithms this requires a huge amount of arithmetic circuitry. Because the 2nd moment depend on the center of gravity the algorithm has to be divided into two steps. In a first step the input data are accepted by Enable++ and stored in a dual-port memory. During this step the center of gravity and the hadronic fraction is calculated. The found center of gravity serves as base pointer for the 2nd moment coefficent tables. In the second step data of the electromagnetic layers are read from the dual-port memory and the 2nd moment calculation is performed. At the same time the 2nd moments are calculated, the center of gravity for the next event can be evaluated.

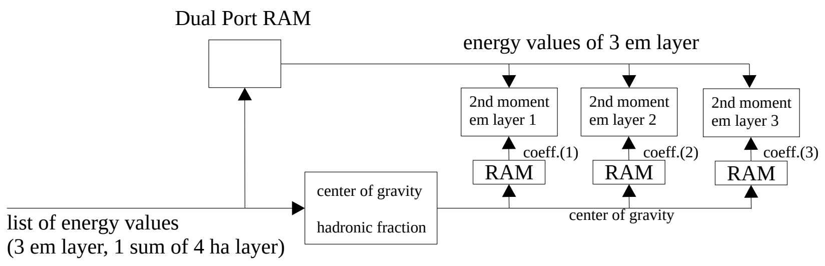

Figure 7: Implementation of the calorimeter algorithm.

Both the center of gravity and the 2nd moment calculation are pipelined and executed at the speed of the incoming data. The two parts of the algorithm require the I/O time of 9.4 μs plus a minor latency of 0.5 μs each. Although the latency of the calorimeter algorithm is 20 μs, the use of the dual-port memory reduces the frequency to 10 μs. The logic resources used for this algorithm consist of four Computing Array FPGAs, which allows the concurrent execution of four ROIs (figure 7).

4.4 Benchmark results

The benchmark results listed in table 1 show that Enable++ is able to handle the high event rate of 100 kHz expected for LHC. Because for some algorithms (TRT and Calorimeter) several ROIs can be executed in parallel on a single FPGA board a rather compact system consisting of two or three crates for the full feature extraction can be designed. Note that the execution time estimations always include the I/O time.

algorithm	Calorimeter	TRT	SCT (2.5%)	SCT (1.0%)
execution time t_e [μs]	10	9	8.5	4
n_{ROI}[9]	4	2	1	1
t_e / n_{ROI} [μs]	2.5	4.5	8.5	4
<speedup>[10]	430	180	580	600

Table 1: L2 trigger benchmark results for Enable++

[9] Number of ROIs implemented on Enable

[10] Mean speedup is the ratio of the mean execution time on three modern RISC processors (Alpha, PowerPC and Sparc10) reported in [1] and on one Enable++ board (considering the

5 Conclusions and Status

The Enable-1 machine and DecPeRLe-1 have proved that FPGA processors are well suited for the high-speed processing required for a data-driven 2nd level trigger in ATLAS. Nevertheless the second generation FPGA processor Enable++ offers several substantial improvement on its predecessors.

- A new feature of Enable++ is the flexible processor topology provided by a computing array configurable by field programmable crossbar switches. In combination with the state-of-the-art FPGA resources and 12 MByte distributed RAM, Enable++ offers strongly enhanced computing power.
- To meet the requirements of different types of subdetectors a major part of Enable++ focuses on a flexible and extremely powerful I/O system. The division of the system in Computing Array Boards and I/O Boards connected by a programmable FPGA bus provides a high degree of scaleability.
- For building new applications the Enable++ Development Environment EDE has been planned and up to now partly realized which consists in the final version of the following tools: A compiler for the high-level hardware description language spC, a simulator and a source level debugger for hardware designs.

Once realized this concept will be an extreme improvement in our FPGA-design capabilities, since it breaks up the design process in small steps which are well defined and independent. This gives programmers freedom in developing complex applications because they can work most of the time on an abstract level without thinking about all pitfalls of hardware design. Debugging of logical errors and first tests can be done with spC and VHDL code while optimisation and hardware debugging are done afterwards. Very fast designs can be implemented using the low-level tools if appropriate. Hardware debugging and system monitoring allows verification of designs for real-time data and real system environments. Together with its inherent scalability Enable++ will be a very powerful, universal and user-friendly FPGA processor system.

All type of boards are under construction and will be operating in a prototype version until summer/fall 1995. An α-release of the spC compiler is already available now and for 9/1995 we scheduled the release of an α-version of the EDE. A full featured stable version of the EDE with complete debugger and simulator support is expected to be available at 6/1996. Thus we are able to take part in the common ATLAS run in fall 1995.

number of ROIs processed in parallel). Note that only for Enable++ the time required for I/O is considered.

References

[BBB$^+$93] J. Badier, R.K. Bock, Ph. Busson, S. Centro, C. Charlot, E.W. Davis, E. Denes, A. Ghorghe, F. Klefenz, W. Krischer, I. Legrand, W. Lourens, P. Malecki, R. Männer, Z. Natkaniec, P. Ni, K.-H. Noffz, G. Odor, D. Pascoli, R. Zoz, A. Sobala, A. Taal, N. Tchamov, A. Thielmann, J. Vermeulen, and G. Vesztergombi. Evaluating Parallel Architectures for two Real-Time Application with 100KHz Repetition Rate. *IEEE Tr. Nucl. Sci.*, 40(1):45–55, 1993.

[BBB$^+$94] D. Belosloudtsev, P. Bertin, R.K. Bock, P. Boucard, V. Dörsing, P. Kammel, S. Khabarov, F. Klefenz, W. Krischer, A. Kugel, L. Lundheim, R. Männer, L. Moll, K.-H. Noffz, A. Reinsch, M. Shand, J. Vuillemin, and R Zoz. Programmable Active Memories in real-time tasks: implementing data-driven triggers for LHC experiments. EAST note 94-27, subm. to Nucl.Inst.Meth.A, 1994.

[BHL94] R.K. Bock, R. Hauser, and I. Legrand. Algorithms in econd-level triggers for ATLAS and benchmark results. ATLAS DAQ note 94-27, EAST note 94-37, 1994.

[BRV93] P. Bertin, D. Rincin, and J. Vuillemin. Programmable Active Memories: A Performance Assessment. Technical report, Digital Equipment Corporation. Paris Research Laboratory, 1993.

[KM94] W. Krischer and L. Moll. Implementation of a pattern recognition algorithm for the Si tracker on DecPeRLe-1. EAST note 94-21, 1994.

[Leg94] I. Legrand. Data collection and preprocessing for the ATLAS second-level trigger. EAST note 94-30, 1994.

[NZK$^+$93] K.-H. Noffz, R. Zoz, A. Kugel, F. Klefenz, and R. Männer. Results of On-Line Tests of the ENABLE Prototype, a 2nd Level Trigger Processor for the TRD of ATLAS/LHC. 1993.