Large-scale, Parallel Embedded Applications: A Hardware Design Model for Software Engineers[1]

M. Fleury, R. P. Self, and A. C. Downton

Department of Electronic Systems Engineering,

University of Essex, Wivenhoe Park,

Colchester, CO4 4SQ, U.K

tel: +44 - 1206 - 872817

fax: +44 - 1206 - 872900

e-mail fleum@essex.ac.uk

[1] A preliminary version of this paper was presented in the form of two talks at the EEUG workshop 2000.

Abstract

Parallel servers are becoming an important sector in the embedded systems marketplace. If software engineers are to implement the multi-algorithm applications that these servers support, then educators should provide clear design routes which inculcate system-level thinking. Pipelined Processor Farms (PPF) is one such top-down design strategy. The contemporary hardware diversity within both processor- and instruction-level parallellism, requires incorporation of a coprocessor model at the node or sub-system layer. Two suitable software-based approaches are reviewed: one which maintains the traditional aspects of hardware modeling, SystemC, and the other, Handel-C, which introduces silicon compilation to the CAD laboratory.

1 Introduction

Developments in the electronic engineering industry require continual updating of university degree courses without merely feeding transitory demands or neglecting enduring principles of design. Presently, thin-client devices, such as PDAs (personal digital assistants)and WAP (Wireless Application Protocol) 'phones, are proliferating in the market place and may eventually far outstrip the volume of conventional PCs. With every thin-client device, a fat server is needed to provide the services which provide product differentiation. The server supports large-scale, embedded applications but is required to give real-time responses, necessitating a parallel architecture.

For example, database enquiries on a mobile 'phone may soon require large-vocabulary speaker-independent continuous speech-recognition servers, as an alternative to conventional call centres. Processor-level parallelism can reduce enquiry latency, and provide sufficient throughput to cope with many simultaneous callers. There may also be a requirement for finer-grained parallelism, where, for example, Viterbi recognition network searches can utilize the streaming SIMD (Single Instruction Multiple Data Stream) instruction sub-set on Intel processors [1].

To compete in this market, companies must cope not only with product lifetimes that can now be shorter than six months, but also with a shortage of software engineers trained for embedded systems; and, more chronically, engineers able to cope with fine-grained parallelism, which requires a hardware orientation. The former implies a stream-lined design process, while the latter implies a training programme for graduate engineers.

At present, many companies within the electronics industry have a short-term strategy of competing for a diminishing supply of hardware-trained graduates with financial inducements, leading to spiraling wage bills. This supply problem is likely to be exacerbated in the short and medium term by the strong preference in the UK (by a ratio of approximately 7:1 [2]) that university applicants show for computer science rather than electronic engineering degree courses. Not only is the supply of hardware-trained graduates strictly limited (about 3,000 admissions to electronics courses per year[1]), but there is a growing fondness for a predominantly software outlook to design. This is evident in the current interest in hardware-software co-design for systems-on-a-chip, and for C/C++ rather than HDL (hardware design language) modeling of hardware. For example, the design of the Neon 256-bit 3D Graphics Accelerator [3] employed C++ templates

[1] Figures are averaged over the last three years.

for bit-width specification, 'C' mathematical libraries, and a fast cycle-accurate simulator written in 'C'. A C-to-Verilog translator enabled hardware synthesis to take place.

Therefore, an obvious question arises: how can universities address the shortfall in electronic engineering personnel equipped to produce hardware/software co-designs and implementations? This paper suggests:

- a generic, top-down design methodology capable of looking at applications as systems;
- a way of accommodating processor-level and instruction level parallelism within the system; and
- a seamless way to pass from high-level design to implementation.

Moreover, the whole should offer clear guidance to the novice hardware designer, and accord with the current software-based orientation. Our attack is two-pronged: (1)an existing top-down design methodology is extended to encompass instruction-level parallelism, and (2)the hardware-specific level of design is removed, in some measure, by means of hardware or silicon compilation.

2 Background

2.1 Top-down design methodology

A system design methodology should offer novice hardware engineers a clearly sign-posted path from a complex application programmed within an over-specified, sequential model of computation to a performance-oriented parallel design. Guidelines for top-level partitioning of the system should be available, together with examples of the system design methodology in action, ideally in a web-based format in the manner of the CalTech Archetypes project [4] for parallel applications. It is also important to include examples of where a methodology cannot be applied. The Pipelined Processor Farm (PPF) methodology [5] has been applied to a variety of contemporary applications in the fields of vision, pattern recognition, and signal processing. It has also been taught in undergraduate courses as a first-cut way to estimate pipeline traversal latency, throughput, and scalability, based on profiling statistics obtained from sequential code.

2.2 Evolving embedded systems

PPF was originally designed to accommodate medium-grained, general-purpose processors, exemplified in the past by the transputer. The original parallel model was of a system built-up in modular fashion from

homogeneous processors. PPFs are pipelines, each stage of which is a scalable data-farm. 'Glueless' multiprocessing, whereby communication link sub-systems are on-chip is still possible, for example in Analog Devices' SHARC series of DSPs [6]. However, reconfigurable hardware with significant, programmable real estate (of the order of 1 million gates in the Xilinx Virtex series of field programmable gate arrays (FPGAs) [7]) can now fulfill the need for finer-grained parallelism at some stage within a processing pipeline. Xilinx also support older FPGA families still suitable for inclusion in mass-produced, consumer products, and well within an educational budget.

As more powerful processors appear, it has also become apparent that the scalability provided by the staple processor farm within the PPF methodology is no longer a critical issue. For example, the processing module within Transtech's Paramid distributed-memory multiprocessor [8] combined a transputer coprocessor with an i860 superscalar computation engine. The performance of a 32-transputer Meiko Computer Surface was outstripped, by an order of magnitude, using a Paramid with just eight modules. Shared-memory, symmetric multiprocessors (SMPs) have also become more common as a parallel solution [9], due to the competition within the database server market. SMPs are only locally scalable [10], due to the finite bandwidth of the bus.

Therefore, in this paper we introduce the concept of Pipelined Parallel Coprocessors (PPC's), in which the pipelined stages may be formed by heterogeneous nodes of limited scalability. A node may also consist of a host commodity processor (RISC or DSP), with support for finer-grained parallelism through the agency of one or more FPGA coprocessors. Of course this is no solution if programming FPGAs is more difficult than programming general-purpose processors.

2.3 Hardware compilation

FPGAs [11] were originally thought of simply as weakly-programmable 'glue' logic, and CAD tools were designed bottom-up to support the hardware design process. However, research into silicon compilers which will seamlessly compile an algorithmic description directly into a layout [12] is coming to fruition, opening up the prospect of a compiler which is compatible with existing software engineering practice, and can integrate into the high-level design processes which commence prior to the coding stage of application development.

Preserving a vertical design process allows a seamless progression [13] from specification to implementation. Lower levels of FPGA design are largely automated so it is in the transition from specification to

HDL program where the weakness lies and where it is possible to iterate between specification and circuit component linkage description (netlist) [14] if 'C' were to be used. All the higher-level parts of an FPGA design might be accomplished by software-oriented engineers with the design constrained by cycle-accurate simulation before passing to the automated parts.

Hardware compilation has the potential to remove an intermediate level of circuit design, represented by hardware languages such as VHDL [15] and Verilog [16], in favour of abstracting away from low-level circuit implementation. To this end we have experimented with both SystemC [17], which uses classes and template libraries within C++, and the Handel-C silicon compiler [18]. Handel-C is a 'C' variant with parallel constructs borrowed from the concurrent programming language occam [19]. Handel-C is strongly-typed, incorporating some of the rigour of occam, which suggests that it is more appropriate for educational use. However, Handel-C does not depart so far from conventional 'C' that porting is not possible. Indeed, it is reported that the complete MP3 audio encoder algorithm has been ported to Handel-C and thence to a dual FPGA implementation [20].

3 PPF route to a design

PPF originated as a simple, unifying idea connecting data-farms in a single pipeline — processor-level parallelism. The data-farm supports dynamic scheduling and incremental scalability. The pipeline allows sequential bottlenecks to be masked, thus escaping a consequence of Amdahl's law (the performance is effectively controlled by the residual sequential content) [21]. PPF has an associated development cycle — transformation of serial code (including legacy code) to parallel form, without changing the algorithms. PPF was first devised in the era of optimism on the future of the modular multicomputer, as a way of countering the effects of Moore's law. However, the arrival of a processor farm on a chip — the TI 'C80 or MVP [22], see Fig. 1, suggested that processor-level parallelism now maps directly down to the hardware.

PPF is a schematic, top-down method of constructing an embedded application, which provides a clear design route to novice engineers. The classic PPF layout is shown in Fig. 2, where three farmer processors are labeled, and where the worker processors are shaded. While worker processors are connected by serial, point-to-point links, the pipeline backplane can be a high-bandwidth bus, thus enabling communication scalability. The regular topology permits replaceable data-farm nodes. Load-balancing is automatic as scheduling is synchronized by returning work (or tokens). Local and inter-stage buffering smoothes data-flow. A limited

number of feed-back loops are permitted along with feed-forward flows. For deterministic workloads, where neither centralized control of scheduling nor a regular topology are needed, then an alternative layout is seen in cascaded processor farms, Fig 3. Synchronization is arranged by means of a data-flow paradigm. The cascaded processor farm solution has the virtue that pipeline traversal latency can be reduced and the farmer is no longer a potential bottleneck.

PPF can be applied to asynchronous and synchronous pipelines alike. Fig. 4 shows an asynchronous, multi-algorithm application, handwritten postcode recognition, that has been partitioned into three processor farms using demand-based farming of data in the classic manner. The third stage is only nominally a farm, as the dictionary search to match classified postcodes against addresses, though parallelizable, was not parallelized in this instance due to the processor granularity (on a Paramid). The H.261 hybrid video encoder, Fig. 5, is the type of complex system now being considered as a system-on-a-chip. Before this can happen, the system has to be analyzed and partitioned. An easily applicable method is to profile the structured, sequential code produced by algorithm and software development teams, so as to expose the relative timings of the component parts, Table 1. In the H.261 encoder case, this allowed a tentative partition of the pipeline to be made, Fig. 6, in which pipeline folding masks synchronization bottlenecks.

PPF is suitable for continuous flow, data-driven (i.e. not control-driven), soft real-time applications, allowing possible throughput, latency, and ordering constraints to be accommodated. It has been used to illustrate top-down design on realistically complex, large-scale embedded systems (5,000–50,000 lines of code). The value of such an approach is that the system is viewed as a whole, before delving into the implementation of a small part of the system. This ensures that resources and effort are not misdirected into optimizing one small part of the application that has little or no impact on overall performance.

4 Extending PPF

PPF is aimed at software-based systems with medium-grained, homogeneous processors. However, an application can contain algorithms within a pipeline, which are better suited to fine-grained parallelism, and which can become a bottleneck. Two examples are: (1) cellular-array algorithms, e.g. relaxation algorithms in optical-flow motion detection suitable for target-tracking, (2) finely synchronous algorithms, e.g. beam-forming of the adaptive- or delay-filter varieties. The problem has been that, in recent times, producers of SIMD fine-grained hardware have not established themselves in the market-place. With the advent of

high-density FPGAs that situation has changed.[2]

When the Karhunen-Loève Transform (KLT) is partitioned to form a PPF the solution is not entirely satisfactory [23] if confined to available medium-grained hardware. The reason Fig. 7 is an 'ill-conditioned' pipeline is that the central Eigenvector calculations, being on a low-dimensional matrix, can not be farmed effectively, but remain as a bottleneck. Due to the large data flows between the two farmers, it is preferable to keep one array or ensemble of images within the same processor, preventing a physical partition of the pipeline. However, the covariance and transform farms are embarassingly parallel, suggesting that FPGAs could be deployed. Therefore, the eigenvector calculations could be mapped to a RISC, while the two transforms could be mapped to FPGA, thus balancing in time the eigenvector calculation with covariance and transform processing. When a coprocessor model is introduced, it thus immediately becomes necessary to partition between the software parts running on the RISC and the hardware parts running on the FPGA.

5 Pipelined examples

This section considers examples of the type of embedded application design for which training is required.

Radar systems [24] provide large-scale, embedded applications that illustrate the problems which will face new engineers. Sample rates are now approaching 500 MHz, decreasing the sample time frame available to give a real-time response. Multidimensional and non-linear signal-processing techniques are being used for 2D and 3D imaging of terrain or objects within Synthetic Aperture Radar (SAR), and target-tracking radar is now required to give estimates of the velocity of moving targets. Algorithm changes within an application are endemic, though there are a few staples such as the FFT. I/O bandwidth may be critical and air-borne systems will require compact, low-power solutions.

A two-level computer model has been proposed [25] that will consist of an invariant network-level structure and a variant node-level structure. In fact, this model only represents the target hardware and it is likely that a third-level is necessary which can emulate the parallel logic present, for the purposes of algorithmic development, and in order to retain a core prototype of the system, which can be projected upon varying target hardware. The parallel structure might be captured within a software component. An embodiment of the two-level computer in given in Fig. 8. The first level consists of a Sparc host with a Myrinet [26] NIC, itself controlled by a LANai communication coprocessor. The host is responsible for initialization of its

[2] There are also plans by Micron Inc. to revive the earlier SIMD DAP coprocessor as a single chip with embedded DRAM.

computation coprocessor and subsequent message handling. The computation coprocessor actually consists of four FPGAs with on-board NIC. The FPGAs are connected to the host by an 8-way crossbar to form what Myrinet call a System Area Network. (The FPGAs can also be connected locally in a ring topology for exchange of global results.) Higher-level connectivity is via Myrinet LAN (160 Mbyte/s in 1998), which replicates, for commodity processors, a form of interconnect common on supercomputers such as the Cray T3D.

Figure 9 shows one node with driver software installed for a particular application, automatic target detection by radar [27]. From Fig. 10, it can be seen where this node fits within the processing pipeline. After data-capture and pre-processing of images by SAR, stage two, Focus of Attention (FOA), is responsible for extracting regions of interest within images with potential targets, e.g. moving vehicles. Second-level detection (SLD), performs simple time-domain template matching, which is embarrassingly parallel. Apart from the use of fine-grained hardware, this application has all of the characteristics of a PPF. It is, in fact, more aptly described as a PPC.

The same processing structure can be applied to sonar beam-forming. However, in detailed studies [28, 29] for time-delay, and frequency-space beam-forming, it was established that there are non-obvious trade-offs between DSPs and FPGAs in terms of, respectively, memory access blockages caused by irregular addressing patterns, and a future bottleneck caused by a limit to the number of i/o pins available on an FPGA. However, the frequency-space implementation runs contrary to the view that silicon compilation will solve many problems, as the CORDIC algorithm, a hardware-oriented solution to calculation of trigonometric values, was needed for manipulation of phase and magnitude components.

Interestingly, an SMP is utilized in an alternative proposal [30] with, apparently, fewer beams but interpolation of samples. A 12 processor Sun Ultra Enterprise achieved a speed-up of eleven with a throughput of 4.8 GFLOP/s. Word-level SIMD parallelism within the SPARC VIS instruction set was used, together with multithreading. In addition, advanced software techniques were applied such as compiler-directed data pre-fetching. Therefore, for optimal designs the software-based approach still needs to go beyond the generic to exploit features of a particular instruction set and microarchitecture.

6 Node-level design

Turning to the design of individual nodes within the PPC, there has been an observable shift towards soft architectures (which are runtime configurable), and away from a fixed hardware architecture, particularly within the design of systems-on-a-chip [31, 32]. In the FPGA world, microprocessors with embedded FPGA coprocessors, such as the Chameleon CS112 [33] are now available. The implication for the designer is that the system model should encompass both software and hardware elements. The design takes on the form of a prototype (an initial core) which, thereafter, is extended with secondary features, but always remains part of the entire system.[3]

The intention behind using C/C++ throughout the modeling/prototyping stage, rather than simply at the initial stage of a design, is that back-end tools will perform the implementation. The result is a productivity gain, because it is easier to change a design at a higher level of abstraction than to change low-level details. Existing software development tools and established environments can be utilized, and legacy code can be ported across into the C/C++ program. However, the C/C++ programming model is sequential, whereas hardware is essentially parallel, implying a need to support concurrency and inter-process communication (IPC). C/C++ also has no timing model, but timing information is required to model hardware clocking. FPGAs have variable-width data-types, and, of course, in general there are a variety of hardware data types, such as tri-state ports, none of which are present in C/C++. On the other hand, other features such as pointers, casting, recursion, and dynamic memory have no counterpart in hardware.[4]

6.1 Extending C/C++

C/C++ is favoured for software-hardware co-design because HDLs do not offer a route to the partition of code between hardware *and* software. Indeed, a design that has previously involved concurrent hardware elements may also take-on a purely software form, requiring software synthesis in order to remove excess

[3] FPGAs are also included in the undergraduate computer engineering curriculum [34] as a means of prototyping ASICs, in a way that mimics Intel and AMD usage. Low-cost boards are employed such as the Altera UPI student CPLD board or Xilinx's XS40 & XS95, or custom boards [35]. There is an educational board with the Xilinx Virtex FPGA intended for laboratory usage. Handel-C is suited to this purpose.

[4] VHDL had its origin as a hardware documentation language, hence the often-heard accusation of verbosity. It is also possible to compare VHDL to C/C++ as a programming language, divorced from the issue of how well hardware is modeled, but this comparison is pursued elsewhere [36].

concurrency [37]. In using an HDL, a design is already committed to hardware, whereas the implementation decision is deferred if the design is held in C/C++ form.

Two main approaches have emerged: (1)class libraries for C++, in the manner of SystemC [38], and (2)language extension of 'C' (and subtractions), exemplified by Handel-C [39]. Handel-C is exclusively targeted at FPGAs of the LUT-based variety, whereas, until recently, SystemC has been intended for ASIC modeling. The authors' relationship to both these products is solely as users.

The key difference between the SystemC and Handel-C development paths, is that the compiler output is respectively an executable specification, and a directly implementable netlist of circuit components. SystemC is compiled with any ANSI C++ compliant compiler, as the libraries needed to execute the model in a simulator are linked in. In SystemC, when porting legacy code from C/C++, there is no need to make modifications. However, further software tools needed to convert the executable specification to synthesized target hardware are not part of the SystemC offering. Therefore, all steps in Fig. 11, which is an abstract representation of the development path as a design flow, beyond production of the 'Executable Model' must go outside SystemC to third-party hardware synthesis and value-added tools. Handel-C's compiler, being proprietary, carries out hardware synthesis internally for a particular FPGA target, and also outputs a simulation file. When coding directly in Handel-C, there is no need to recode to produce a synthesizable subset. However, Table 2 [18] summarizes the syntactic differences between Handel-C and conventional 'C', which must be accounted for when passing to and from hardware. In the Handel-C design flow, Fig. 12, all steps beyond 'Data Refinement' can be performed within the Handel-C environment.

Though the design flows of Figs. 11 and 12 have a superficial resemblance, in fact the superstructure provided by SystemC is more aligned to behavioural modeling stage in an HDL. A sign of the emphasis on modeling is that the simulator in SystemC is integrated with the binaries, thus improving simulation speed. Iterating through implementation refinement in Handel-C involves re-coding a 'C' program in order to reduce the number of cycles. It is only when the code is transferred to the FPGA, that it becomes apparent that the clock width may also need to be reduced. However, increasingly due to short product lifetimes, the need to refine or 'tweak' VHDL is also less pressing. Note also that it is perfectly possible to arrange for VHDL output from Handel-C.

The refinement process in both SystemC and Handel-C currently continues to require a design expert as knowledge of how to improve the hardware/code resides in the designer.

6.2 Hardware compilation with SystemC

SystemC is an industry-led *de facto* standard, which addresses the need for multi-vendor tool inter-operability. The founding members of the consortium have each contributed expertise (intellectual property) in the form of class libraries, as follows:

- *Synopsis*: 'Scenic' modeling environment;
- *CoWare*: module interface abstraction technology;
- *Frontier Design*: fixed-point data-type mapping.

The class libraries and simulator are supplied on a no-fee basis.

Both System-C and Handel-C have been applied to coding a simple counter circuit, Fig. 13, [17].

The SystemC version of the code is given in Fig. 14. It will be seen that the nomenclature and conceptual model of HDLs such as VHDL and Verilog has been preserved in SystemC. For example, SystemC has modules, ports, signals, and clocks. In the figure, the `module` macro is simply expanded to a C++ `struct` (a class with no private members). C++ template classes allow ports to be parameterized by data-type. Within the `main` control section (not shown), signals and clocks linking the ports of modules are similarly instantiated. In the figure, a process is created through a C++ constructor, through the `SC_CTOR` macro. Associated with the process is a sensitivity list, as in VHDL, the definition being inherited from an `sc_module` superclass. Therefore, processes are registered with the SystemC kernel (linked in later), and are activated whenever an event, such as a positive clock edge, triggers it. In terms of modeling concurrency, SystemC also provides thread processes which are implemented as co-routines (self-scheduling processes [40]), which conveniently avoids having to create a separate method for each state of a module. A thread continually executes unless it suspends itself or passes control to another thread.

SystemC can be employed to model a variety of hardware behaviour. For example, a clocked thread is associated with a clock which steps the thread through the actions of a finite state machine. Clocked threads can model bus behaviour. SystemC clocks, specified in terms of time units, duty cycle, and start-up settings, can be applied (within testbenches) to regions of a circuit.

SystemC can also model communication in an implementation neutral manner (asynchronous and unclocked), through an RPC (remote procedure call) mechanism. The called procedure is a method in another module, which notionally executes in zero time.

6.3 Silicon compilation with Handel-C

Handel-C is the product of Celoxica Ltd.[5] who have produced an industry-strength version of the research language Handel [41, 42]. However, like Handel, Handel-C has wrapped parallel constructs from `occam` within a 'C' syntax. Therefore, Handel-C is not a 'C' wrapper for an HDL, which brings the advantage of occam's simplicity, but also the disadvantages of unfamiliarity and possibly incompatibility.

Unlike `occam`'s asynchronous timing model, Handel-C is clock synchronous, with all assignment statements assumed to take a unit clock cycle. The main parallel construct is `par` which runs constituent sets of statements (processes) in parallel, and in lock-step. A process cannot leave a `par` block until all processes have completed in the manner of the `cobegin` construct [43]. Procedures are currently implemented by macros, which take untyped parameters, passed by reference. A prioritized alternation construct is also adapted from `occam`, allowing inputs (and outputs). IPC is by means of channels, which synchronize through `rendezvous` semantics. Handel-C channels (asynchronous but clocked) are the sole form of IPC. In Fig. 15, the only other data types apart from channels are `char` and various `ints`, which have an associated data width.

Handel-C is in essence a silicon compiler, mapping a high-level language onto hardware, a result of a long research programme [44, 45]. Unlike the familiar compiler targeting general-purpose hardware, a silicon compiler targets reconfigurable hardware. Occam-like constructs are directly implemented as silicon in predefined combinational logic, which carries the risk of inefficient implementations in terms of gate usage and timing, compared to hand-crafted circuits. In our experience [36], the opposite is possible. A more significant problem may be the single-clock simulation model as even on FPGAs there are a variety of internal and external clocks. As the difference in treatment of clocks illustrates, in contrast to the conservative approach of SystemC, Handel-C has replaced hardware modeling by software with software modeling by hardware.

7 Conclusion

The Pipelined Processor Farms (PPF) methodology is particularly suitable for mapping large-scale applications, written using structured or object-oriented programming, onto partitioned, asynchronous pipelines. Parallelism can be introduced within each partitioned stage in the form of a data farm. PPF's stylized

[5] Celoxica Ltd. formerly traded as Embedded Solutions Ltd.

methodology, though by no means a panacea, does have the virtue of providing a first cut design, rather than leave the engineer searching for an optimal design which, with today's rapid product turnover, would be counter-productive. Individual algorithms, which in the past were mapped onto medium-grained, general-purpose processors by necessity, can now be transferred to reconfigurable, programmable hardware, chiefly in the form of RISC/FPGA combinations. However, this is an unwelcome development because of a dwindling supply of engineers conversant with hardware issues.

Fortunately, CAD tool developers have moved towards making the 'look-and-feel' of modeling tools more 'C'-like and less HDL-like. Below the outward appearance, educators are presented with a choice: SystemC hardware compilation models concurrency by coroutines, interprocess communication by signals, and hardware timing by local clocks, much within the conceptual framework familiar to VHDL or Verilog-trained engineers; while Handel-C models concurrency by cobegin blocks, interprocess communication by channels, and hardware timing by a local clock, in a manner familiar to parallel programmers of `occam` and the transputer. Handel-C does not model hardware as such, but presents (through silicon compilation) a software model to hardware. Alternatively, Handel-C can be compared to Java, though rather than the Java Virtual Machine, we have the synchronous `occam` virtual machine.

The simplicity of the Handel-C virtual machine may, like occam before it, limit Handel-C's applicability, whereas SystemC is suitable for modeling ASICs. For educators, this is probably not a significant disadvantage as FPGAs are commonly employed to model other digital architectures. One practical consideration is that Handel-C provides a 'one-stop' solution, whereas SystemC is dependent on forthcoming third-party hardware synthesis tools. If the objective is to train software-oriented engineers to participate in the application system design process then Handel-C does indeed abstract away from the hardware, which should increase its acceptability to Computer Science students.

Acknowledgement

This work is being carried out with assistance from an EPSRC/DERA CASE studentship no. 9930329X.

References

[1] A. C. Valles. Using streaming SIMD extensions to boost speech recognition performance. Technical report, Intel Inc., 2000. Available as `http://developer.intel.com/update/archive/issue19/stories/ss1.htm`.

[2] Universities and Colleges Admissions Service (UCAS). Summary statistics for 1997-1999. Technical report, 2000. Figures available from `http://www.ucas.com/figures/archive/download/index.html`.

[3] J. McCormack, R. McNamara, C. Gianos, N. P. Jouppi, T. Dutton, J. Zurawski, L. Seiler, and K. Correll. Implementing the Neon: A 256-bit graphics accelerator. *IEEE Micro*, 19(2):58–69, 1999.

[4] K. M. Chandy. Concurrent program archetypes. In *Scalable Parallel Libraries Conference*, pages 1–9. IEEE Computer Society, Los Alamitos, CA, 1994.

[5] M. Fleury and A. C. Downton. *Pipelined Processor Farms: Structured Design for Parallel Embedded Systems*. Wiley, New York, 2001. In press.

[6] P. Graham and B. Nelson. Reconfigurable processors for high-performance, embedded digital signal processing. In *Field Programmable Logic and Applications, FPL'99*, pages 1–10. Springer, Berlin, 1999. LNCS No. 1673.

[7] Xilinx Inc., 2100 Logic Drive, San Jose, CA. *Virtex 2.5V Field Programmable Gate Arrays Datasheet*, 2000. Available as `http://www.xilinx.com/partinfo/d2003.pdf`.

[8] Transtech Parallel Systems Group, 20 Thornwood Drive, Ithaca, NY. *The Paramid User's Guide*, 1993.

[9] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[10] R. Duncan. Developments in scalable MIMD architectures. In A. Zomaya, editor, *Parallel Computing: Paradigms and Applications*, pages 3–24. Thomson, London, 1996.

[11] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vransic. *Field-Programmable Gate Arrays*. Kluwer, Boston, 1992.

[12] B. M. Pangrle and D. D. Gajski. Design tools for intelligent silicon compilation. *IEEE Transactions on Computer-Aided Design*, 6(6):1098–1112, 1987.

[13] I. Page. Constructing hardware-software systems from a single description. *Journal of VLSI Signal Processing*, 12:87–107, 1996.

[14] J. F. Wakerly. *Digital Design: Principles and Practices.* Prentice Hall, Upper Saddle River, NJ, 3rd edition, 2000.

[15] K. C. Chang. *Digital Systems Design with VHDL and Synthesis.* IEEE Computer Society, Piscataway, NJ, 1999.

[16] S. Palnitkar. *Verilog HDL: A Guide to Digital Design and Synthesis.* Sunsoft, Mountain View, CA, 1996.

[17] Synopsys Inc. & Open SystemC Initiative. *SystemC v. 1.1 User's Guide*, 2000. Available from `http://www.systemc.org`.

[18] M. Bowen. *Handel-C Language Reference Manual, 2.1.* Embedded Solutions Ltd, Abingdon, UK, 1998.

[19] *occam 2 Reference Model.* Prentice Hall, New York, 1988.

[20] Converting mp3 software to hardware, 2000. Application note available at `http://www.embeddedSol.com/`.

[21] G. Amdahl. Validity of single processor approach to achieving large scale computing capabilities. In *Proceedings of the Spring Joint Computer Conference*, pages 483–485. AFIPS, 1967.

[22] K. Balmer, N. Ing-Simmons, P. Moyse, I. Robertson, J. Keay, M. Hammes, E. Oakland, R. Simpson, G. Barr, and D. Roskell. A single chip multimedia video processor. In *IEEE Custom Integrated Circuits Conference*, pages 91–94, 1994.

[23] M. Fleury, A. C. Downton, and A. F. Clark. Karhunen-Loève transform: An exercise in simple image-processing parallel pipelines. *Computers and Artificial Intelligence*, 19(1):19–36, 2000.

[24] L. H. J. Bierens. Hardware design for modern radar processing. *Electronics & Communication Engineering*, pages 257–270, December 1997.

[25] C. Seitz. A prototype two-level multicomputer, 1996. Project information on DARPA/ITO project available from `http://www.myri.com/research/darpa/96summary.html`.

[26] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W-K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, pages 29–36, 1995.

[27] W-K. Su, R. Siviloti, Y. Cho, and D. Cohen. Scalable, network-connected, reconfigurable, hardware accelerators for an automatic-target-recognition application. Technical report, Myricom Inc., Arcadia, CA, 1998.

[28] P. Graham and B. Nelson. FPGAs and DSPs for sonar processing—inner loop computations. Technical report, Configurable Computing Lab., Electrical and Computer Eng. Dept., Brigham Young Univ., 1998. Technical Report No. CCL-1998-GN-1.

[29] P. Graham and B. Nelson. Frequency-domain sonar processing in FPGAs and DSPs. In *IEEE Symposium on FPGAs for Custom Computing Machines, FCCM'98*, 1998.

[30] G. E. Allen and B. L. Evans. Real-time sonar beamforming on a Unix workstation using process networks and PTHREADS. *IEEE Transactions on Signal Processing*, 48(3):921–926, 2000.

[31] J. Rabaey, A. Abnous, Y. Ichimawa, K. Seno, and M. Wan. Heterogeneous reconfigurable systems. In *IEEE Workshop on Signal Processing Systems, SIPS 97*, pages 24–34, 1997.

[32] W. K. Gass. System integration issues for set-top boxes. In *IEEE Workshop on Signal Processing Systems, SIPS 97*, pages 65–75, 1997.

[33] X. Tang, M. Aalsma, and R. Jou. A compiler directed approach to hiding configuration latency in Chameleon processors. In *Field-Programmable Logica and Applications, FPL 2000*, pages 29–38. Springer, Berlin, 2000. Lecture Notes in Computer Science No. 1896.

[34] J. O. Hamblen. Rapid prototyping using field programmable logic devices. *IEEE Micro*, 20(3):29–37, 2000.

[35] A. Zemva, A. Trost, and B. Zajc. Educational programmable system for prototyping digital circuits. *International Journal of Electrical Engineering Education*, 35(3):236–244, 1998.

[36] M. Fleury, R. P. Self, and A. C. Downton. Hardware compilation for software engineers: an ATM example. *IEE Proceedings in Software*, 2001. Accepted for publication.

[37] B. Lin. Efficient compilation of process-based concurrent programs without run-time scheduling. In *Design Automation and Test, DATE'98*, 1998.

[38] J. Gerlach and W. Rosenstiel. System level design using the SystemC modeling platform. In *SDL'2000*, 2000. Available from `http://www.systemc.org/technical_papers.htm`.

[39] D. Šulik, M. Vasilko, D. Ďuračkova, and P. Fuchs. Design of a RISC microcontroller core in 48 hours. In *Embedded Systems Show, London Olympia 2000*, 2000. Available from `http://dec.bournemouth.ac.uk/dec_ind/decind6/drhw/publications.html`.

[40] A. Burns and A. Wellings. *Real-time Systems and their Programming Languages*. Addison-Wesley, Wokingham, UK, 1989.

[41] I. Page and W. Luk. Compiling occam into FPGAs. In W. R. Moore and W. Luk, editors, *FPGAs*, pages 271–283. EE & CS, Abingdon, UK, 1991.

[42] M. Spivey, I. Page, and W. Luk. *How to program in Handel*. Oxford University Hardware Compilation Unit, 1995.

[43] P. Brinch-Hansen. The design of Edison. *Software Practice and Experience*, 11(4):362–396, 1981.

[44] P. Hilfinger. A high-level language and silicon compiler for digital signal processing. In *IEEE Custom Integrated Circuit Conference*, pages 213–216, 1985.

[45] D. May. Compiling occam into silicon. In C. A. R. Hoare, editor, *Developments in Concurrency and Communication*, pages 87–129. Addison-Wesley, Reading, MA, 1990.

Table 1: H.261 Execution Time Profile

Execution Sequence	Function Name	Per frame Exec. Time (Normalized)
1	clear_picture	0.147
2	copy_field	0.196
3	read_picture_frame	1.200
4	copy_macro_block_data	0.728
5	encoder_motion_estimation_h261	24.306
6	encoder_decisions_h261	6.392
7	forward_transform_picture	2.230
8	h261_forward_quantize_picture	1.398
9	tm_forward_scan_picture	0.632
10	forward_run_level_picture	0.712
11	h261_code_picture	0.591
12	inverse_run_level_picture	0.486
13	tm_inverse_scan_change_picture	0.634
14	h261_inverse_quantize_picture	0.991
15	inverse_transform_picture	2.232
16	reconstruct_h261	2.813
17	macro_block_to_line	0.531
18	tm_calculate_snr	0.753
19	tm_display_statistics	0.134
20	write_picture_file	1.217

Table 2: Reserved Word Comparison Between 'C' and Handel-C

Types, Type Operators, Objects		
In Both	In 'C' Only	In Handel-C
int	double	chan
unsigned	float	ram
char	enum	rom
long	register	chanin
short	static	chanout
	extern	undefined
	struct	interface
	volatile	
	void	
	const	
	union	
Statements		
In Both	In 'C' Only	In Handel-C
{;}	continue	par
switch	return	delay
do ... while	goto	?
while	typedef	!
if ... else		prialt
for(;;)		
break		

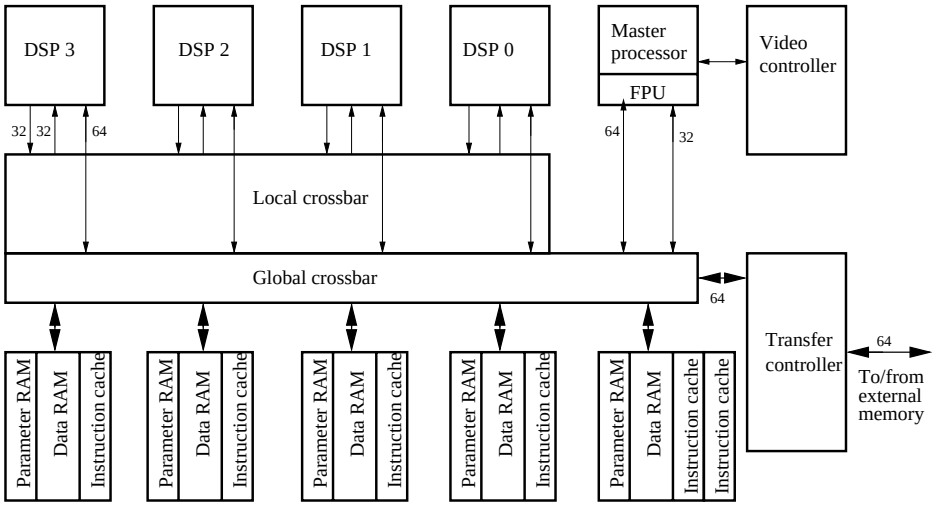

Figure 1: TI's MVP architecture.

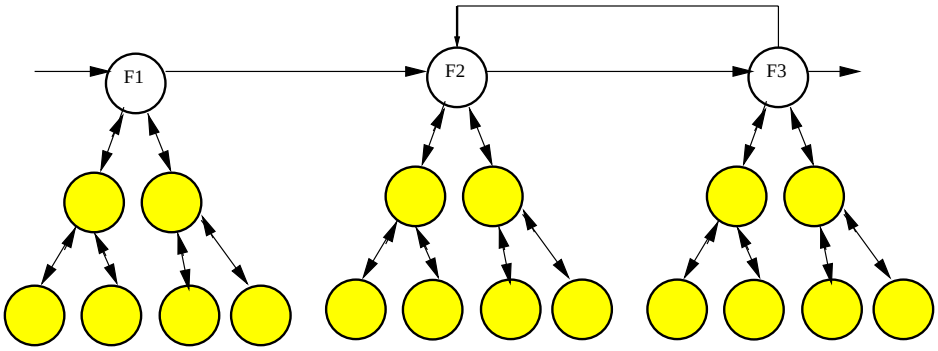

Figure 2: Classic PPF layout.

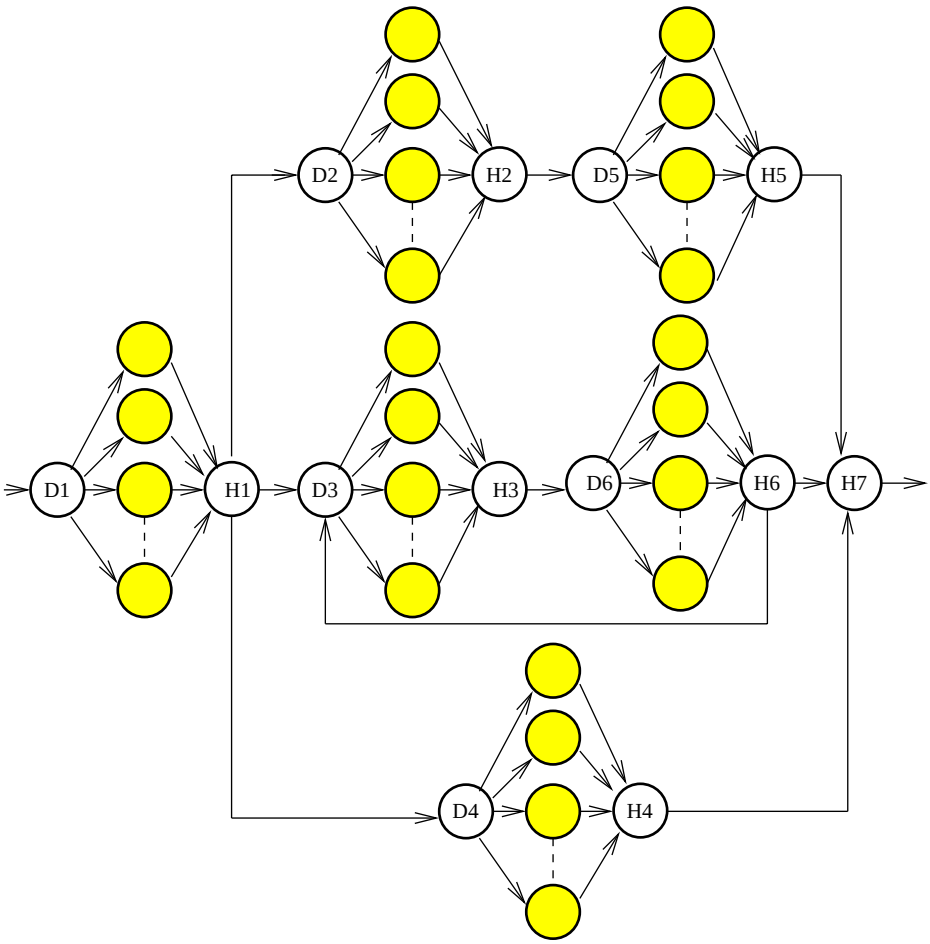

Figure 3: Cascaded processor farms.

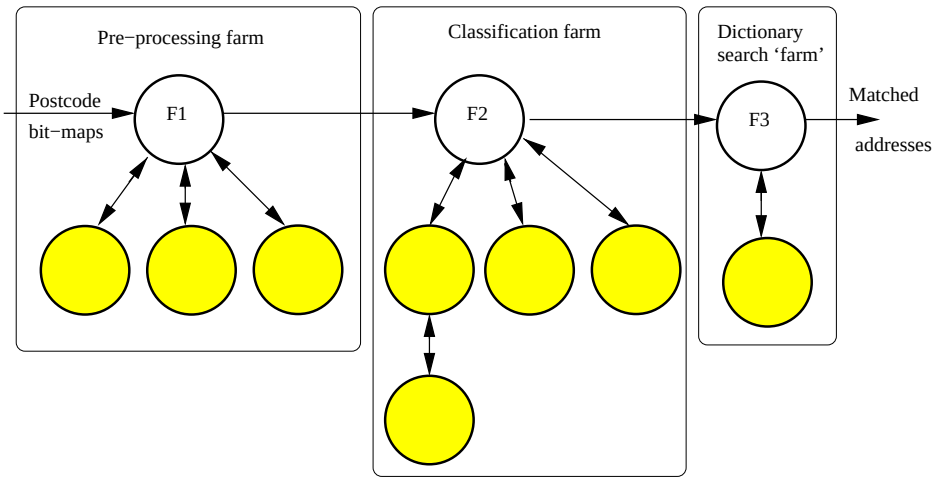

Figure 4: Asynchronous PPF example: Handwritten postcode recognition pipeline.

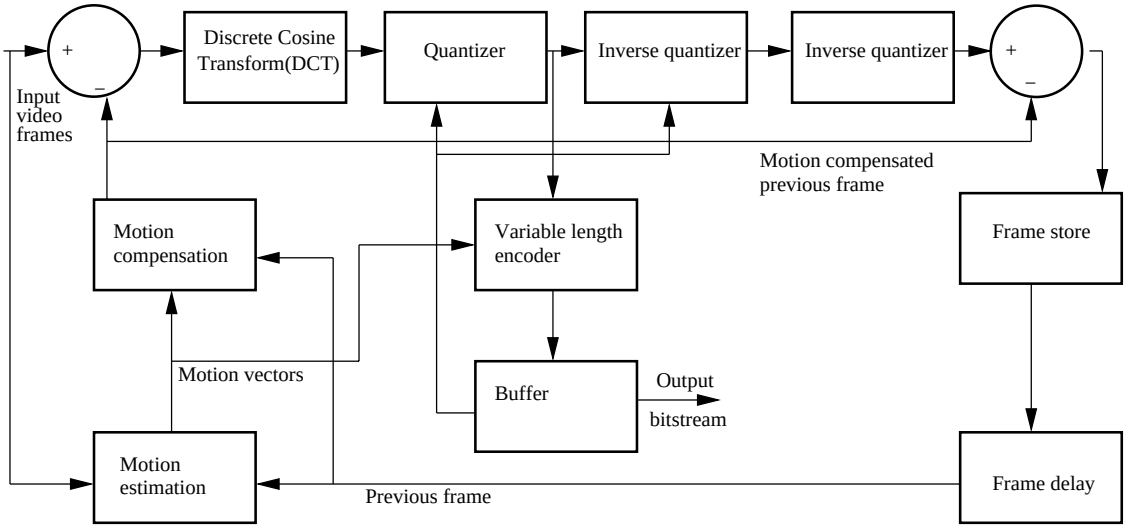

Figure 5: Synchronous PPF example: H.261 encoded block diagram.

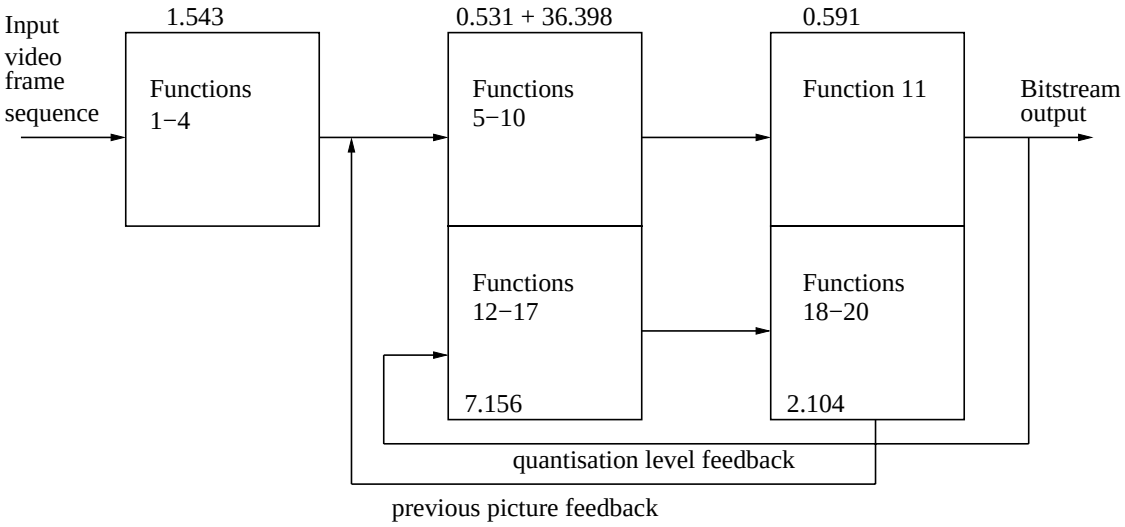

Figure 6: Partition of H.261 into a PPF.

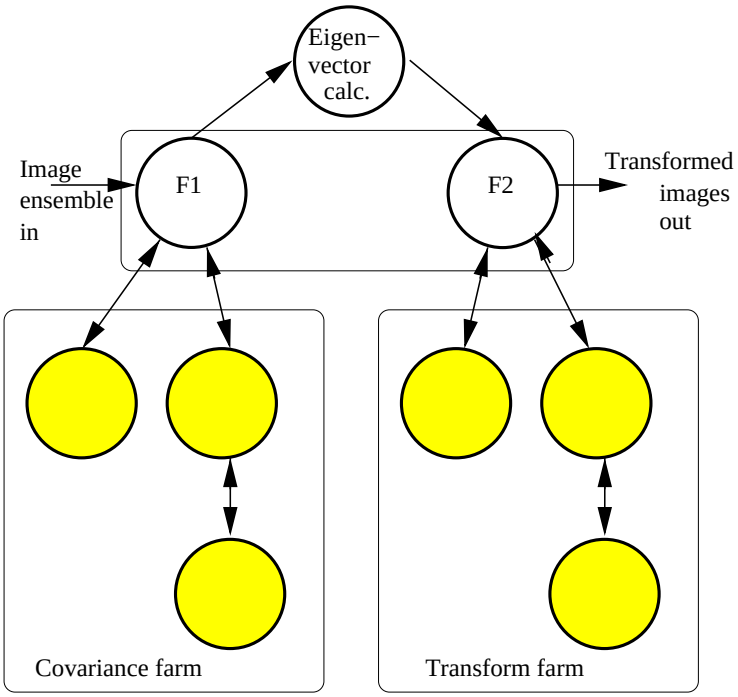

Figure 7: KLT 'ill-conditioned' pipeline.

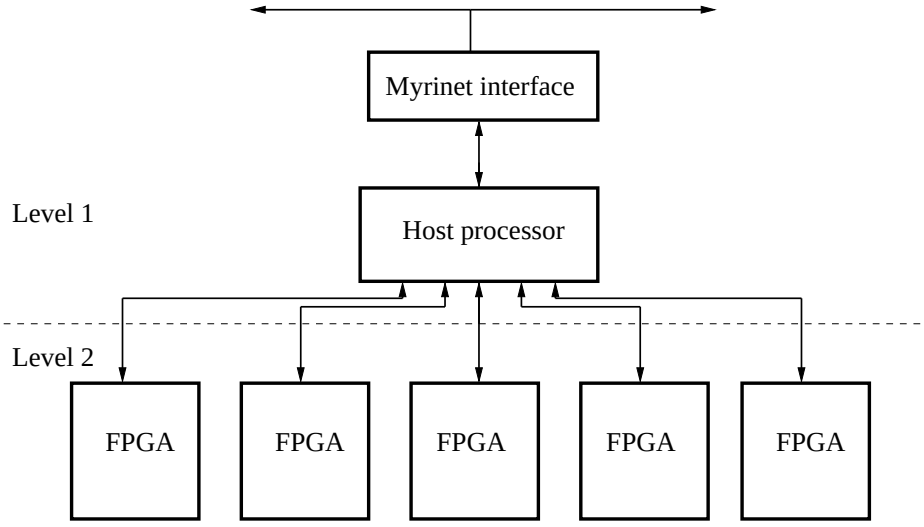

Figure 8: Two-level computer example.

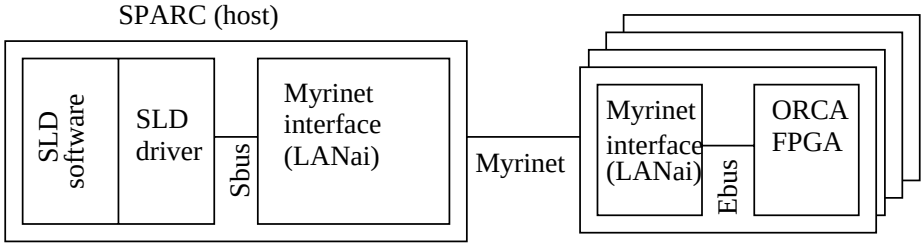

Figure 9: Second-level detection node.

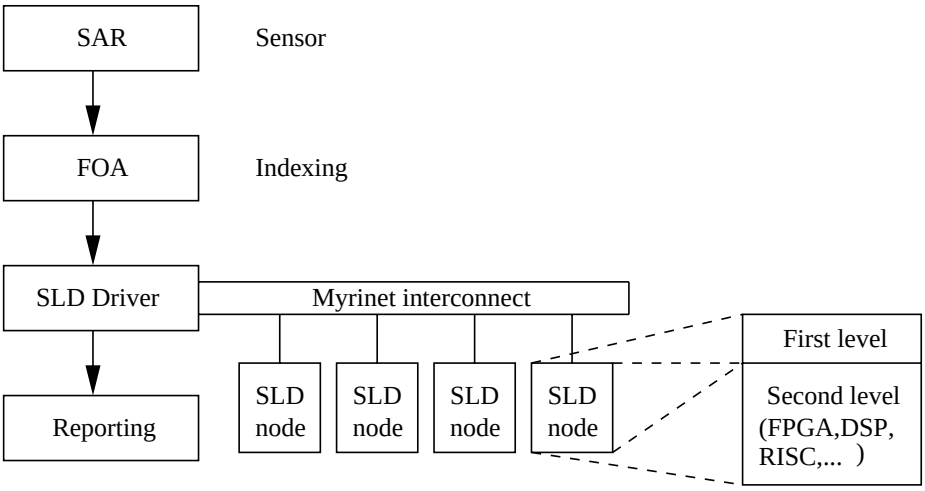

Figure 10: ATR pipeline.

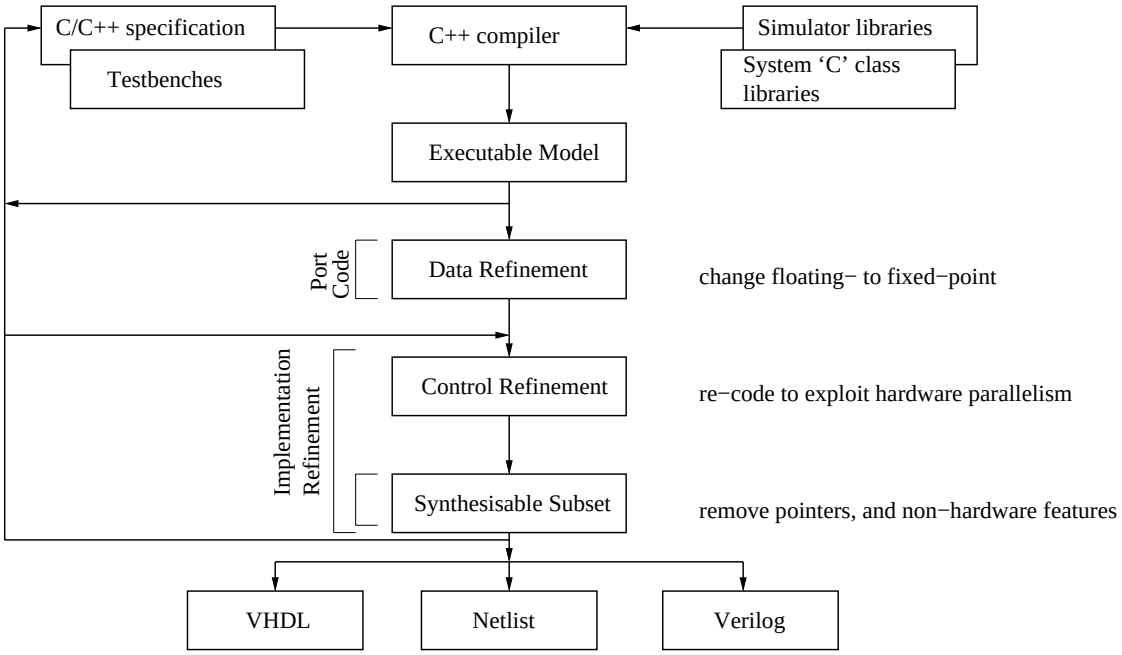

Figure 11: SystemC design flow.

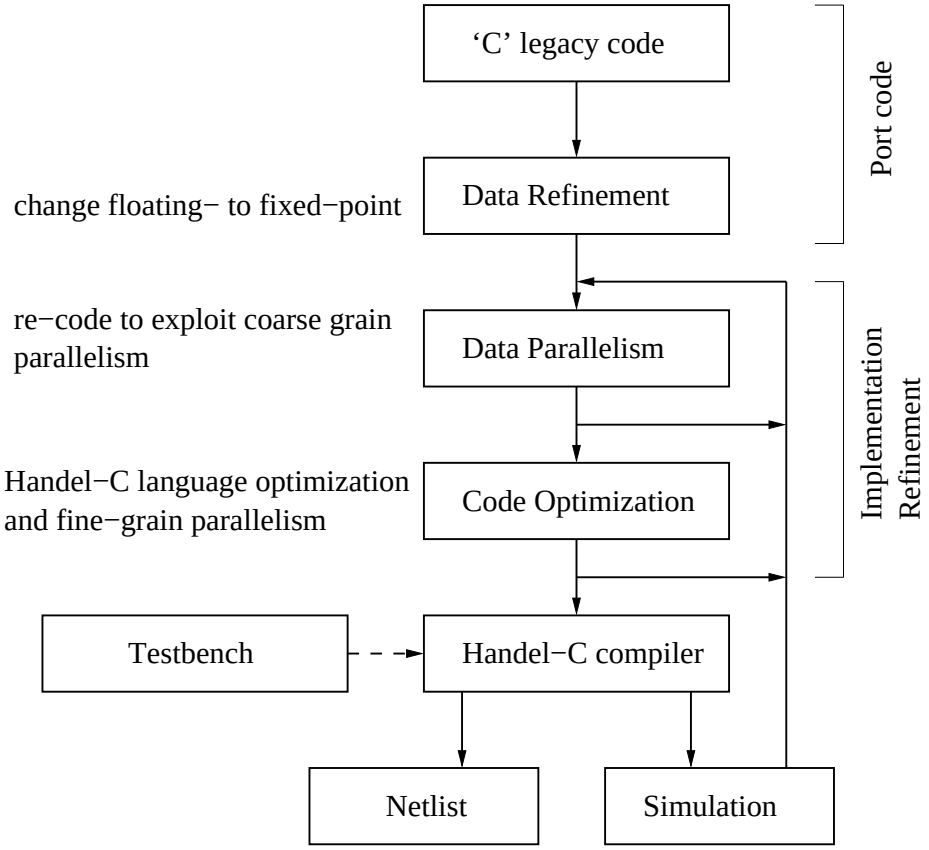

Figure 12: Handel-C design flow.

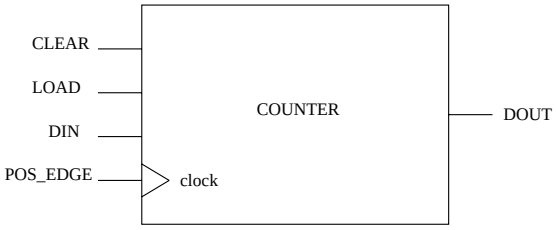

Figure 13: Simple counter module.

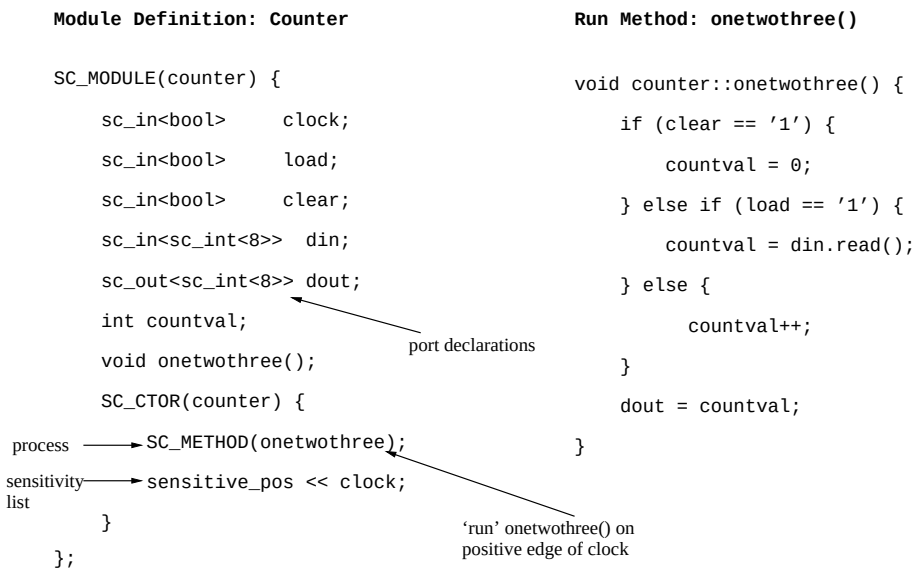

Figure 14: SystemC code version of 'Counter'.

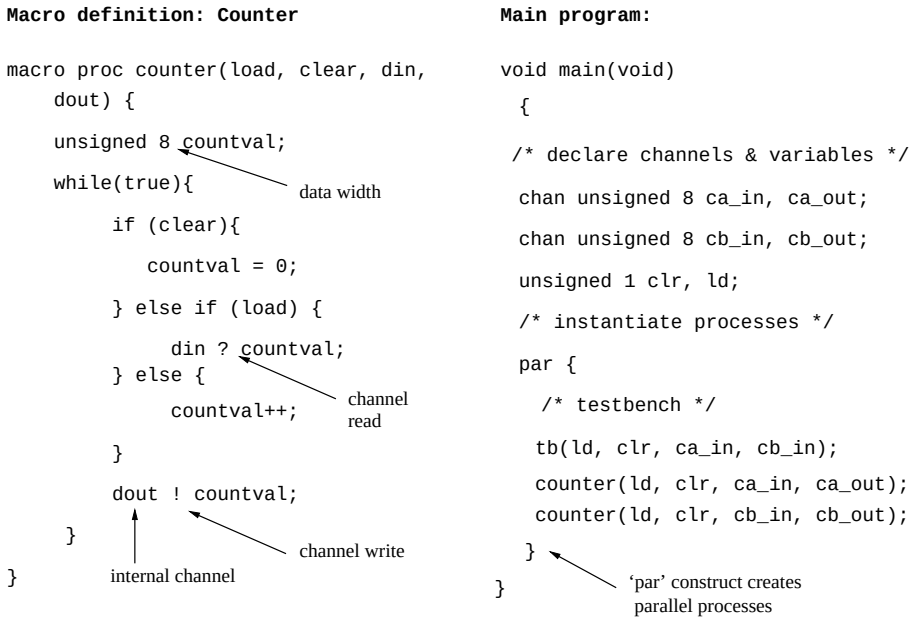

Figure 15: Handel-C code version of 'Counter'.