The PowerPC Backend Molen Compiler

Elena Moscu Panainte, Koen Bertels, and Stamatis Vassiliadis

Computer Engineering Lab
Delft University of Technology, The Netherlands
{E.Panainte, K.Bertels, S.Vassiliadis}@et.tudelft.nl

Abstract. In this paper, we report on the backend C compiler developed to target the Virtex II Pro PowerPC processor and to incorporate the Molen architecture programming paradigm. To verify the compiler, we used the multimedia video frame M-JPEG encoder of which the Discrete Cosine Transform (DCT*) function was mapped on the FPGA. We obtained an overall speedup of 2.5 against a maximal theoretical speedup of 2.96. The performance efficiency of 84 % is achieved using automatically generated but non-optimized DCT* hardware implementation.

1 Introduction

Reconfigurable computing (RC) is becoming increasingly popular as it bears the promise of combining the flexibility of software with the performance of hardware. Some concern can be expressed because the current state of the art assumes that the developer has a deep understanding of both software and hardware development before the benefits of this technology can be exploited. This justified concern underlines the necessity to intensify research and development efforts to support the designer in this process. The Delft Workbench is an initiative that investigates the integration and development of tools supporting the different design phases starting at code profiling, synthesis and ending at the generation of binary code. The idea is to automate as much as possible the design exploration and the final development process. This paper addresses an important part of the tool chain, namely the construction of a backend compiler that targets such a hybrid platform. The compiler allows on the basis of function annotations, the automatic modification of applications to generate the appropriate binaries.

The current paper reports on the completed compiler targeting the Molen implementation on the Virtex II Pro platform FPGA. The contributions of the paper can be summarized as follows :

- A compiler backend targeting the PowerPC processor included in the Molen prototype has been developed.
- The theoretical compiler extensions presented in [1] have been implemented and adjusted to the target Field-programmable Custom Computing Machine (FCCM) features.
- Software/hardware development tools have been integrated to automatize the design flow phases.

The application that was used for experiments is M-JPEG encoder. Measurements show that the resulting code executed on the implementation of the Molen organization on the Virtex II Pro board, allows to obtain overall speedups of 2.5 when compared to the software only execution. We emphasize that the goal of this experiment is not to study multimedia extensions but rather to provide a proof of concept of the compiler toolset targeting FCCMs. We also stress that in contrast to the work discussed in [1], the presented paper bases all experimentation on a real Molen prototype rather than estimations.

The paper is organized as follows. In the next section, we present the Molen organization and discuss related work. In section 3, we present the compiler extensions required for the PowerPC processor and the Molen prototype. We then present the case study where the computation intensive DCT function is mapped on the reconfigurable fabric and show that speedups of 2.5 are achieved.

2 Background and Related Work

In this section, we briefly discuss the Molen programming paradigm [2], describe the Molen machine organization that supports it and discuss related work.

The Molen programming paradigm [2] is a sequential consistency paradigm for programming FCCMs possibly including a general-purpose computational engine(s). The paradigm allows for parallel and concurrent hardware execution and is intended (currently) for single program execution. For a given ISA, a one time architectural extension (based on the co-processor architectural paradigm) comprising 4 instructions (for the minimal πISA as defined in [2]) suffices to provide an almost arbitrary number of operations that can be performed on the reconfigurable hardware. The four basic instructions needed are **set**, **execute**, **movtx** and **movfx**. By implementing the first two instructions (**set/execute**) an hardware implementation can be loaded and executed in the reconfigurable processor. The **movtx** and **movfx** instructions are needed to provide the communications between the reconfigurable hardware and the general-purpose processor (GPP). The Molen machine organization [3] that supports the Molen programming paradigm is described in Figure 1. The two main components in the Molen machine organization are the 'Core Processor', which is a GPP and the 'Reconfigurable Processor' (RP). Instructions are issued to either processors by the 'Arbiter' by means of a partial decoding of the instructions received from the instruction fetch unit. The support for the SET/EXEC instructions required in the Molen programming paradigm is based on *reconfigurable microcode*. The reconfigurable microcode is used to emulate both the configuration of the Custom Computing Unit (CCU) and the execution of implementations configured on the CCU. A detailed description of how the Molen organization and programming paradigm compare with other approaches is presented in [1].

An overview of research that aims to combine GPPs and reconfigurable hardware and to provide software support for programming these FCCMs and a discussion of how they relate to research reported in this paper includes the following:

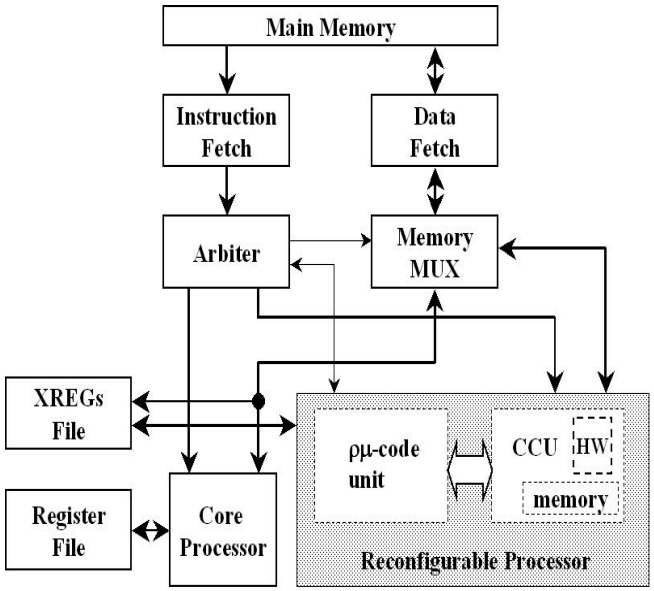

Fig. 1. The Molen machine organization

Reconfigurable Architectures Performance: Several reconfigurable architectures have been proposed in the last decade (see [4] for a classification). The reported performance improvements are mainly based on simulation (see for example [5]) or estimation (e.g. [6] [7]) results. Eventhough some implementations exist [8], in most cases the performance is just estimated. In this paper, we present the validation of the Molen approach based on a real and running implementation of the Molen reconfigurable processor platform.

Compilers for Reconfigurable architectures: When targeting hybrid architectures to improve performance, the applications must be partitioned in such a way that certain computation intensive kernels are mapped on the reconfigurable hardware. Such mapping is not simple as it assumes deep understanding of both software and hardware design. Several approaches (e.g. [5] [7]) use standard compilers to compile the applications to FCCMs. As standard compilers do not target reconfigurable architectures, the kernel computations implemented in hardware are manually replaced by the appropriate instructions for communication with and controlling the reconfigurable hardware. This replacement is done manually and it is a time-consuming [9] and error-prone process. In order to facilitate the design and development process, much effort is put in the development of automated tools (compilers) to perform such tasks [10] [6] [11]. However, the extensions of the cited compilers mainly aim to generate the instructions for the reconfigurable hardware and they are not designed to easily support new optimizations that exploit the possibilities of the reconfigurable hardware. The Molen compiler presented in this paper, is based on a a flexible and extensible infrastructure that allows to add easily new optimization and analysis passes that take into account the new features of the target reconfigurable architecture.

Compiler Support for Hiding the Reconfiguration Latency: One of the major drawbacks of the reconfigurable hardware is the the huge reconfiguration latency [12] [4]. Different techniques such as configuration caching and prefetching (e.g. [3]) have been proposed to reduce the reconfiguration latency. These hardware techniques should be combined with compiler optimizations that provide an efficient instruction scheduling to use the available parallelism between different FCCMs components in the hardware reconfiguration phase. Nevertheless, many FCCMs do not expose a specific instruction for hardware reconfiguration (see [4] for FCCMs classification), thus impeding compiler support for hiding reconfiguration latency. We schedule the SET instruction (which performs the hardware configuration) as early as possible from the hardware execution phase, resulting in exploiting the parallelism between the GPP and the FPGA during the configuration stage.

3 The Molen Compiler

The Molen compiler comprises the Stanford SUIF2[13] (Stanford University Intermediate Format) Compiler Infrastructure for the front-end and the Harvard Machine SUIF framework[14] for developing compiler backends and optimization. In [1], the theoretical compiler extensions target a virtual Molen reconfigurable architecture including an *x86* processor as a GPP. In this section we present the implemented compiler backend and extensions required for the Molen hardware implementation on the Virtex II Pro platform FPGA which includes a PowerPC processor.

PowerPC Backend: The first step is to have a backend C-compiler that generates the appropriate binaries to be executed on the PowerPC processor integrated on the Virtex II Pro board. Current MachineSUIF backends excluded the backend for PowerPC architecture. In consequence, we developed a PowerPC compiler backend and implemented the PowerPC instruction generation, PowerPC register allocation, PowerPC EABI stack frame allocation and software floating-point emulation (not completed). Additionally, in order to exploit the opportunities offered by the reconfigurable hardware, the PowerPC backend has to be extended in several directions, as described in the rest of this section.

Hiding Configuration Latency: Due to the lack of support for dynamic reconfiguration in the current Molen implementation (there was not sufficient information about the Virtex II Pro platform) and taking into account that in our experiments there is only one function (DCT*) executed on the reconfigurable hardware, the Molen compiler generates in advance only one SET instruction for DCT* at the application entry point. The SET instruction does not stall the GPP implying that the compiler can issue this instruction as far ahead as possible from the hardware execution phase. This scheduling allows the GPP to execute in parallel with the FPGA during the configuration stage. This is particularly useful for the cases when the SET instruction is initially included in a loop. Thus, issuing the SET instruction at the application entry point avoids unnecessary repetitive hardware configuration. The cases when multiple operations are

sequentially executed on the reconfigurable hardware and do not simultaneously fit on the target FPGA are not covered by this scheduling.

Compiler Extensions for Molen Implementation: First of all, a general purpose reconfigurable architectural scheme (presented in [3]) has been adopted. We implemented the minimal instruction set extension, containing the following:

- SET/EXECUTE instructions are included in the MIR (Medium-level Intermediate Representation) and LIR (Low-level Intermediate Representation) of the Molen compiler. In order not to modify the PowerPC assembler and linker, the compiler generates the instructions in binary form. For example, for the instruction *exec 0x80000C* the generated code is *.long 1A000031* where the encoding format (presented in [15]) is recognized by the arbiter.
- MOVTX/MOVFX: The equivalent PowerPC instructions are *mtdct/mfdcr*. Moreover, the XRs (exchange registers) are not physical registers but they are mapped at fixed memory addresses.

```
la 3, 12016(1)          #load the address of the first param
la 12, 12016(1)         #load the address of the second param
mtdcr 0x00000056,3      #send the address of the first parameter
mtdcr 0x00000057,12     #send the address of the second parameter
sync                    #
nop                     #synchronization
nop                     #
nop                     #
bl main._label0         #instr. required by the arbiter impl.
main._label0:
.long 0x1A000031        #exec 0x8000C
nop                     #synchronization
```

Fig. 2. Code generated by the Molen compiler for the reconfigurable DCT* execution

In Figure 2, we present the code generated by the Molen compiler for the DCT* function call executed on the reconfigurable hardware. In order to correctly generate the instructions for hardware configuration and execution, the compiler needs information about the DCT* hardware implementation. This information is described in an FPGA Description File, which contains for the DCT* operation the fields presented in Figure 3. Line 2 defines the start memory address from where the XRs are mapped. In line 3, the compiler is informed that there is a hardware implementation for the DCT* operation with the microcode addresses for SET/EXECUTE instructions defined in lines 4-5. The *sync* instruction from Figure 2 is a PowerPC instruction that ensures that all instructions preceding *sync* in program order complete before *sync* completes. The sequences of *sync* and *nop* instruction are used to flush the processor pipeline. The SET instruction is not included in the above example because it has been scheduled earlier by the Molen compiler previously presented.

```
1: NO_XRS        = 512      # number of available XRs
2: START_XR      = 0x56     # the memory address of the first XR
3: OP_NAME       = dct      # info about the DCT* operation
4: SET_ADDR      = 0x39A100 # the address of the DCT* SET microcode
5: EXEC_ADDR     = 0x80000C # the address of the DCT* EXEC microcode
6: END_OP                   # end of the info about the DCT* operation
.....                       # info about other operations
```

Fig. 3. Example of an FPGA Description File

4 M-JPEG Case Study

In this case study we report the performance improvements of the Molen implementation on the Virtex II Pro for the multimedia video frame M-JPEG encoder.

Design Flow: The design flow used in our experiments is depicted in Figure 4. In the target application written in C, the software developer introduces pragma annotations for the functions implemented on the reconfigurable hardware. These functions are translated to Matlab and processed by the COMPAAN[16]/LAURA[17] toolchain to automatically generate the VHDL code. The commercial tools can then be used to synthesize and map the VHDL code on the target FPGA. The application is compiled with the Molen compiler and the executable is loaded and executed on the target Molen FCCM.

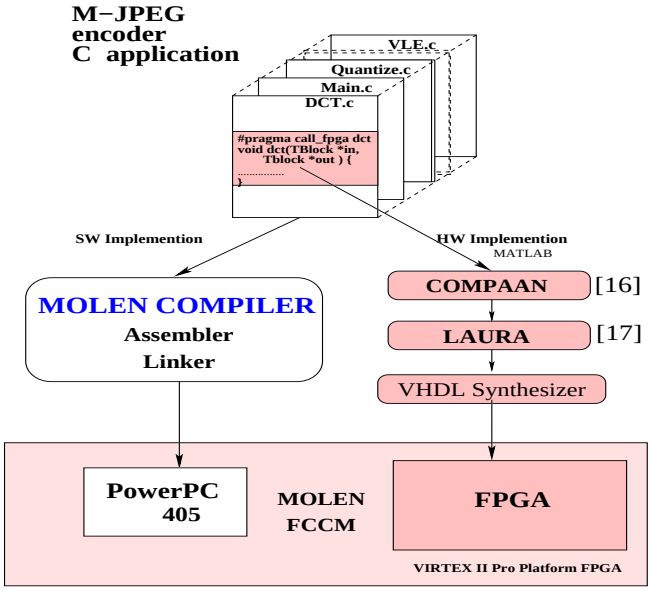

Fig. 4. The design flow

M-JPEG, Software and Hardware Implementations: The application domain of these experiments is the video data compressing. We consider a real-life application namely Motion JPEG (M-JPEG) encoder which compresses sequences of video frames applying JPEG compression for each frame. The in-

put video-frames used in these experiments are presented in Table 1. The M-JPEG implementation is based on the public domain implementation described in "PVRG-JPEG CODEC 1.1", Portable Video Research Group, Stanford University. The most demanding function in M-JPEG application is 2D DCT with preshift and bound transforms (named in this paper as DCT*). In consequence, DCT* is the first function candidate for hardware implementation. The only modification of the M-JPEG application that indicates the reconfigurable DCT* execution is the introduction of the pragma annotation as presented in Figure 4. The hardware implementation for execution of the DCT* function on the reconfigurable hardware is described in [9]. The VHDL code is automatically extracted from the DCT* application code using COMPAAN[16]/LAURA[17] tools. The Xilinx IP core for DCT and the ISE Foundation[18] are used to synthesize and map the VHDL code on the FPGA. After the whole application is compiled with the Molen compiler described in the previous section, in the final step we use the GNU assembler and linker and the C libraries included in the Embedded Development Kit (EDK) [19] from Xilinx to generate the application binary files. The target FCCM is the implementation of the Molen reconfigurable architecture on the Virtex II Pro platform FPGA of Xilinx described in [15]. In this implementation, the GPP is the IBM PowerPC 405 processor immersed into the FPGA fabric.

Name	# frames	Resolution [pixels]	Format	Color/BW
tennis	8	48x48	YUV	color
barbara	1	48x48	YUV	color
artemis	1	48x48	YUV	color

Table 1. M-JPEG video sequences

Performance Measurements: The current Molen implementation is a prototype version, which imposes the following constraints:

- the memory size for *text* and *data* sections are limited to maximum 64K. In order for the M-JPEG executable to fulfill these limitations, we rewrote the original application preserving only the essential features for compressing sequences of video frames. Moreover, these limitations also restrict the size of the input video frames to 48x48 pixels (Table 1, column 3).
- dynamic reconfiguration is not supported (yet) on the Molen prototype. In consequence, we could not measured the impact on performance of repetitive hardware configurations.

In addition, the performance measurements have been performed given the following additional conditions:

- the input/output operations are extremely expensive for the current Molen prototype, due to the standard serial connection implemented by UART at 38400 bps between the Molen prototype and the external environment; this

limitation can be removed by the implementation of faster I/O system. We therefore did not include the I/O operation impact in our measurements as they are not relevant for RC paradigm

- the DCT* hardware implementation requires a different format for the input data than the software implementation. Consequently, an additional data format conversion is performed in software before and after the DCT* execution on reconfigurable hardware.
- taking into account that the target PowerPC processor included in the Virtex-II Pro platform does not provide hardware floating-point support and that the required floating-point software emulation is extremely expensive, the integer DCT is used for both software and hardware implementation to allow a fair comparison.

The execution cycles for M-JPEG encoder and comparisons are presented in Table 2. As we considered a sequence of 8 video frames for *tennis* input sequence, we present only the minimal and maximal values for each measurement in order to avoid redundant information.

Pure Software Execution: In Table 2(a), we present the results of our measurements performed on the the Virtex II Pro platform, when the M-JPEG application is entirely executed on the PowerPC processor. In row 1, the number of cycles used for executing the whole M-JPEG application is given. In row 2, the cycles consumed by one execution of the DCT* function are given and the next row contains the total number of cycles spent in DCT*. From these numbers, we can conclude that 66% of the total execution time is spent in the DCT* function, given the input set. This 66% represents the maximum (theoretical) improvement that can be obtained by hardware acceleration of the DCT* function. The corresponding theoretical speedup - using *Amdahl's law* - is presented in Table 2(c), row 2.

Execution on the Molen prototype: In Table 2(b), we present the number of cycles for the M-JPEG execution on the Molen prototype. From row 1 we can conclude that an overall speedup of 2.5 (Table 2(c), row 1) is achieved. The DCT* execution on the reconfigurable hardware takes 4125 cycles (row 2) which is around 300 times less than the software based execution on the PowerPC processor (Table 2(a), row 2). However, due to the data format conversion required by the DCT* hardware implementation, the overall number of cycles for one DCT* execution becomes 102,589 (Table 2(b), row 3), resulting in a 10 fold speedup for DCT* and a 2.5 speedup globally. The performance efficiency is about 84% as presented in Table 2(c), last column. It is noted that this efficiency is achieved even though i) the hardware implementation has been automatically but non-optimally obtained (using COMPAAN[16]/LAURA[17] tools) and ii) additional software data conversion diminished the DCT* speedup in hardware. From these measurements, we can conclude that even non-optimized implementation can be used to achieve considerable performance improvements. In addition, taking into account that only one function (DCT*) is executed on the reconfigurable hardware, we consider that an overall M-JPEG speedup of

		tennis [0-7]		barbara	artemis
		MIN	MAX		
Pure Software Execution (a)	M-JPEG	33,671,821	33,779,813	34,014,157	34,107,893
	DCT*	1,242,017	1,242,017	1,242,017	1,242,017
	DCT* cumulated	22,356,306	22,356,306	22,356,306	22,356,306
	Maximal improvement	66.18%	66.39%	65.73%	65.55%
Execution on Molen prototype (b)	M-JPEG	13,443,269	13,512,981	13,764,509	13,839,757
	DCT* HW	4,125	4,125	4,125	4,125
	DCT* HW + Format conv.	102,589	102,589	102,589	102,589
Comparison (c)	Practical speedup	2.50	2.51	2.47	2.46
	Theoretical speedup	2.96	2.98	2.92	2.90
	Efficiency	84.17%	84.65%	84.70%	84.91%

Table 2. M-JPEG execution cycles and comparisons

2.5x from the theoretical speedup of 2.96 x confirm the viability of the presented approach.

5 Conclusions

In this paper, we presented the implemented compiler support for the Molen implementation on the Virtex II platform FPGA. The compiler allows the automatic modification of the application source code using the extensions following the Molen Programming Paradigm. The experiment evaluated the effectively realized speedup of reconfigurable hardware execution of the DCT* function of the M-JPEG application. The generated code was executed on the Molen prototype and showed a 2.5 speedup. This speedup consumed 84% of the total achievable speedup which amounts to 2.9. Taking into account that hardly any optimization was performed and only one function ran on the reconfigurable fabric, a significant performance improvement was nevertheless observed. We emphasize that we do not compare the RC paradigm to other approaches for multimedia applications boosting performance (such as MMX, 3DNow!, SSE). The focus of this paper was rather on the compiler support for the Molen FCCM under the RC paradigm. Further research on the compiler will address optimizations for dynamic configurations and parallel execution on the reconfigurable hardware.

References

1. Moscu Panainte, E., Bertels, K., Vassiliadis, S.: Compiling for the Molen Programming Paradigm. In: 13th International Conference on Field Programmable Logic and Applications (FPL). Volume 2778., Lisbon, Portugal, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2003) 900–910

2. Vassiliadis, S., Gaydadjiev, G., Bertels, K., Moscu Panainte, E.: The Molen Programming Paradigm. In: Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation, Samos, Greece (2003) 1–7
3. Vassiliadis, S., Wong, S., Cotofana, S.: The MOLEN $\rho\mu$-Coded Processor. In: 11th International Conference on Field Programmable Logic and Applications (FPL). Volume 2147., Belfast, UK, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2001) 275–285
4. Sima, M., Vassiliadis, S., S.Cotofana, van Eijndhoven, J., Vissers, K.: Field-Programmable Custom Computing Machines - A Taxonomy. In: 12th International Conference on Field Programmable Logic and Applications (FPL). Volume 2438., Montpellier, France, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2002) 79–88
5. Campi, F., Cappelli, A., Guerrieri, R., Lodi, A., Toma, M., Rosa, A.L., Lavagno, L., Passerone, C.: A reconfigurable processor architecture and software development environment for embedded systems. In: Proceedings of Parallel and Distributed Processing Symposium, Nice, France (2003) 171–178
6. Kastrup, B., Bink, A., Hoogerbrugge, J.: Concise: A compiler-driven cpld-based instruction set accelerator. In: Proceedings of FCCM'99, Napa Valley CA (1999) 92–100
7. Rosa, A.L., Lavagno, L., Passerone, C.: Hardware/Software Design Space Exploration for a Reconfigurable Processor. In: Proc. of DATE 2003, Munich, Germany (2003) 570–575
8. Lee, M.H., Singh, H., Lu, G., Bagherzadeh, N., Kurdahi, F.J.: Design and Implementation of the MorphoSys Reconfigurable Computing Processor. VLSI Signal Processing Systems **24** (2000) 147–164
9. Stefanov, T., Zissulescu, C., Turjan, A., Kienhuis, B., Deprettere, E.: System Design using Kahn Process Networks: The Compaan/Laura Approach. In: Proc. of DATE 2004, Paris, France (2004) 340–345
10. Gokhale, M.B., Stone, J.M.: Napa C: Compiling for a Hybrid RISC/FPGA Architecture. In: Proceedings of FCCM'98, Napa Valley, CA (1998) 126–137
11. Ye, Z.A., Shenoy, N., Banerjee, P.: A C Compiler for a Processor with a Reconfigurable Functional Unit. In: ACM/SIGDA Symposium on FPGAs, Monterey, California, USA (2000) 95–100
12. M. Bolotski, A. DeHon, Knight, J.T.F.: Unifying FPGAs and SIMD arrays. In: ACM/SIGDA Symposium on FPGAs, Berkeley, CA (1994) 1–10
13. (http://suif.stanford.edu/suif/suif2)
14. (http://www.eecs.hardvard.edu/hube/research/machsuif.html)
15. Kuzmanov, G., Vassiliadis, S.: Arbitrating Instructions in an $\rho\mu$-coded CCM. In: Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL'03). Volume 2778., Lisbon, Portugal, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2003) 81–90
16. Kienhuis, B., Rijpkema, E., Deprettere, E.: Compaan: Deriving Process Networks from Matlab for Embedded Signal Processing Architectures. In: Proc. of CODES'2000, San Diego, CA (2000) 13–17
17. Zissulescu, C., Stefanov, T., Kienhuis, B., Deprettere, E.: Laura: Leiden Architecture Research and Exploration Tool. In: 13th International Conference on Field Programmable Logic and Applications (FPL). Volume 2778., Lisbon, Portugal, Springer-Verlag Lecture Notes in Computer Science (LNCS) (2003) 911–920
18. (http://www.xilinx.com/ise_eval/index.htm)
19. (http://www.xilinx.com/ise/embedded/edk.htm)