An Overview of a Compiler for Mapping MATLAB Programs onto FPGAs

P. Banerjee
Department of Electrical and Computer Engineering
Northwestern University
2145 Sheridan Road, Evanston, IL-60208
banerjee@ece.northwestern.edu

Abstract

This paper describes a behavioral synthesis tool called the MATCH compiler developed as part of the DARPA Adaptive Computing Systems program. The MATCH compiler reads in high-level descriptions of DSP applications written in MATLAB, and automatically generates synthesizable RTL models in VHDL. The RTL models can be synthesized using commercial logic synthesis tools and place and route tools onto FPGAs. By linking the two design domains of DSP and FPGA hardware design, the MATCH compiler provides DSP design teams a significant reduction in design labor and time, elimination of misinterpretations and costly design rework, automatic verification of the hardware implementation, and the ability of systems engineers and algorithm developers to perform architectural exploration in the early phases of their development cycle. The paper describes how powerful directives are used to provide high-level architectural tradeoffs for the DSP designer. The MATCH compiler has been transferred to a startup company called AccelChip which has developed a commercial version of the compiler called AccelFPGA. Experimental results are reported using AccelFPGA on a set of nine MATLAB benchmarks that are mapped onto the recent Xilinx Virtex II and Altera Stratix FPGAs. The benchmark programs range in complexity from 20 lines to 170 lines of MATLAB code and produce VHDL code ranging from 1500 to 4500 lines of code. The compilation times range from 3 seconds to 40 seconds.

1. Introduction

The performance requirements of today's communication systems, such as 3G and 4G wireless communication systems, MPEG4 video and Video over IP, now exceed the capabilities of general-purpose processors. With the introduction of advanced Field-Programmable Gate Array (FPGA) architectures such as the Xilinx Virtex-II [14], and the Altera Stratix [2], a new hardware alternative is available for DSP designers that combines all the benefits of general-purpose processors with the performance advantage of ASICs.

DSP design has traditionally been divided into two types of activities – systems/algorithm development and hardware/software implementation. The majority of DSP system designers and algorithm developers use the MATLAB language [9]. The first step in this flow is the conversion of the floating point MATLAB algorithm, into a fixed point version using quantizers from the Filter Design and Analysis (FDA) Toolbox for MATLAB. Algorithmic tradeoffs such as the precision of filter coefficients and the number of taps used in a filter are performed at the MATLAB level. Hardware design teams take the specifications created by the systems engineers and algorithm developers (in the form of a fixed point MATLAB code) and create a physical implementation of the DSP design. If the target is an FPGA, PLD or ASIC, the first task is to create a register transfer level (RTL) model in a hardware description language (HDL) such as VHDL and Verilog. The RTL HDL is synthesized by a logic synthesis tool, and placed and routed onto an FPGA using backend tools. The process of creating an RTL model and a simulation testbench takes about one to two months with the tools currently used today.

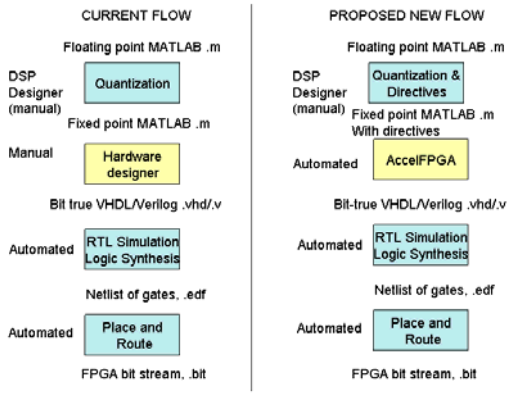

Figure 1. Automated design using MATCH

This paper described the MATCH compiler developed at Northwestern University under sponsorship of the DARPA Adaptive Computing Systems program. The MATCH compiler reads in fixed point MATLAB behavioral models and automatically outputs synthesizable RTL models in VHDL. The resultant RTL VHDL is bit-true with the original fixed point MATLAB specification. The current manual and new automated flow is shown in Figure 1. MATCH also allows users to perform quick iterations of hardware designs, allowing area and speed trade offs and architecture exploration.

2. Related Work

The problem of translating a high-level or behavioral language description into a register transfer level representation is called high-level synthesis [6]. Synopsys developed one of the first successful commercial behavioral synthesis tools in the industry, the Behavioral Compiler [12], which took behavioral VHDL or Verilog and generated RTL VHDL or Verilog. Recently, there has been a lot of work in the use of the C programming language and other high-level languages to generate synthesizable HDL codes or hardware implementations [7,10]. There have been several commercial efforts to develop compilers taking C/C++ into VHDL or Verilog. Examples are CoWare, Adelante [1], Celoxica [3], C Level Design [4] and Cynapps [5]. SystemC is a new language developed by the SystemC consortium which allows users to write hardware system descriptions in a language similar to C++ [11]. Synopsys has a tool called Cocentric which takes SystemC and generates RTL VHDL/Verilog.

While there are some companies which develop related tools from C or C++ to VHDL and Verilog, this paper describes the MATCH compiler [8] from Northwestern University that takes behavioral MATLAB descriptions (the default language of DSP design) and generates RTL VHDL for FPGA design. Some of the unique and challenging features of the MATLAB language are the support for array operations (operating on matrices instead of scalars), an interpretive environment where the types and shapes of variables are not declared at compile time but inferred at runtime, and a very powerful set of built in library functions.

3. Directives in MATCH Compiler

MATCH compiler directives are used to bridge the gap between the MATLAB source and the synthesis of the computational structures created by MATCH. The most important role of the directives is for the user to provide the tool domain specific knowledge as well as opportunities of many optimizations. Every compiler directive is prefixed by "%!ACCEL". This makes the directives appear as comments to other environments dealing with MATLAB since all comments in MATLAB start with %. Some of these directives are described in more detail below.

3.1. TARGET Directive

By specifying the %!MATCH TARGET XC2V250 directive, the compiler becomes aware of the characteristics of that target Virtex II architecture, namely that it can support 1536 Combinational Logic Blocks, 48 Kbits of distributed RAM, 24 embedded multipliers, 24 embedded RAM blocks. By specifying the %!MATCH TARGET EP1S10 directive, the compiler becomes aware of the characteristics of the Altera Stratix architecture, namely that it consists of 94 M512 RAM Blocks, 60 M4K RAMs, 1 MegaRAM Blocks, 6 DSP Blocks, and a 40 X 30 array of Logic Array Blocks.

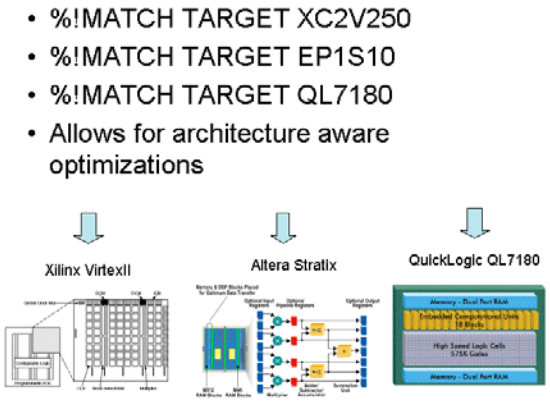

Figure 2. Illustration of the TARGET directive.

3.2. BEGIN_HARDWARE Directive

MATCH allows the user to use hardware partitioning directive to demarcate parts of the input source that are targeted for hardware synthesis and parts that are not. The BEGIN_HARDWARE and END_HARDWARE directives indicate a section of MATLAB code that is intended for hardware synthesis.

3.3. SHAPE Directive

MATLAB is designed to be an interpretive environment where the types and shapes of variables are determined at run time. For the purpose of compiling MATLAB programs to hardware in order to generate VHDL or Verilog, it is necessary to determine the types and shapes of all variables at compile time. The shape directives are used to convey necessary shape information with respect to the variables that appear within the hardware section of

the source code that is being mapped into hardware. For example, %!MATCH SHAPE a(30,40,50) defines a 3-dimensional array 'a' with 30, 40, and 50 elements in the corresponding dimensions. The shape information is required for array variables that appear in the hardware section (for MATLAB programs enclosed within the BEGIN_HARDWARE and END_HARDWARE directives). The compiler attempts to infer the shapes of many dependent variables.

3.3. STREAM Directive

The purpose of the STREAM directive is the specification of the type of data flow that inputs and outputs of the synthesized hardware will handle. Streaming data is defined as data with a regular rate of flow through the hardware. For systems that will handle streaming data, MATCH supports the automatic creation of ports with the required buffering mechanisms to sustain the regular flow of data with the use of the STREAM directive. These mechanisms include 'double-buffering' to allow concurrent processing of data and buffering of new data samples. The syntax of the STREAM directive is as follows.

```
%!MATCH STREAM <stream-index>
for stream-index = STARTVAL:STRIDE:ENDVAL
        BEGIN_HARDWARE indata
        in_buf = indata(stream-index);
        …
        outdata(stream-index) = out_buf;
        END_HARDWARE outdata
end
```

where stream-index is the index in the control statement of a 'for' loop

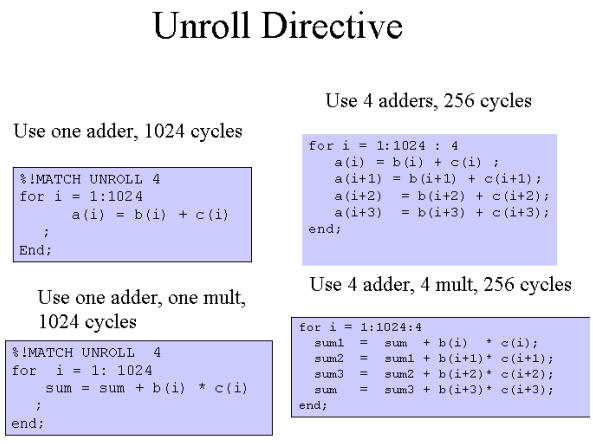

Figure 3. Illustration of the UNROLL directive.

3.4. UNROLL Directive

The UNROLL directive is a mechanism to expand the source MATLAB to create more copies of loop bodies, thereby increasing performance optimizations. Let us consider an example MATLAB for loop.

```
%!MATCH UNROLL 4
for i = 1: 16
   sum = sum + b(i) * c(i) ;
end;
```

Without the UNROLL directive, the MATLAB code has one addition and one multiplication operation in the data flow graph of its basic block hence the MATCH compiler will generate an RTL VHDL or Verilog which will use one adder and one multiplier to schedule this computation which will take 16 cycles. If the code were to be unrolled as shown, the loop body will be replicated four times and the loop index in successive copies are incremented. In addition, scalars that carry values from one iteration to another iteration are renamed. For example, the scalar "sum" would be renamed in successive copies. This exposes opportunities to chain operations to the compiler.

```
for i = 1:4:16
  sum1  =  sum  + b(i)  * c(i);
  sum2  =  sum1 + b(i+1)* c(i+1);
  sum3  =  sum2 + b(i+2)* c(i+2);
  sum   =  sum3 + b(i+3)* c(i+3);
end;
```

MATCH now recognizes four addition and four multiplication operations in each basic block hence it will schedule it across four cycles using four adders and four multipliers in parallel. The UNROLL directive is therefore used by the user to generate different area-delay hardware alternatives. It is illustrated in Figure 3.

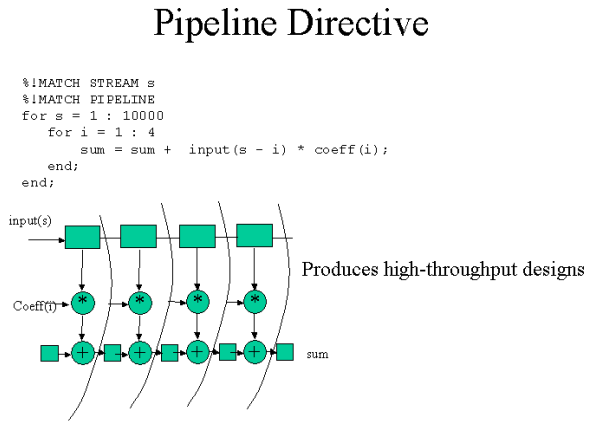

Figure 4. Illustration of the PIPELINE directive.

3.5. PIPELINE Directive

Pipelining increases the throughput of a datapath by introducing registers in the datapath. This increase in throughput is particularly important when the datapath is iterated in the overall design. The PIPELINE directive is placed just before the loop, whose body is to be pipelined. For pipelining function bodies the directive is placed just above the function definition.

```
%!MATCH   PIPELINE
for i =  init:  end
    for body ……..
end;

%!MATCH  PIPELINE
function    x = foo( y )
   function body …..
end;
```

The PIPELINE directive is illustrated in Figure 4.

4. Results on Benchmarks

The MATCH compiler has been commercialized by a company called AccelChip [15] in a product called AccelFPGA

We now report some experimental results on various benchmark MATLAB programs using the AccelFPGA compiler.

- A 16 tap Finite Impulse Response Filter
- A 64 tap memory mapped tiled FIR filter
- A Decimation in Time FIR filter
- A 64 point Fast Fourier Transform
- A Least Mean Square adaptive LMS filter
- An Infinite Impulse Response Filter of type DF1
- An Interpolation FIR filter
- A Block Matching Algorithm
- A Digital Subscriber Line (DSL) algorithm

Table 1 shows some benchmark characteristics of the MATLAB programs. It can be seen that the MATLAB programs vary in size from 20 lines to 175 lines with an average length of 60. We also show the number of directives used in the 9 benchmark programs. The corresponding synthesizable RTL Verilog versions of the designs are quite large, varying in size from 883 lines to 4188 lines with an average length of 2265. We also include the compile times of AccelFPGA version 1.4 for each of the benchmarks. All execution times were measured on a Dell Latitude Model C610 laptop with a 1.2GHz Pentium III CPU, 512 MB RAM, and 80 GB hard drive running Windows 2000. It can be seen that the execution time varies from 2.5 seconds to 39 seconds. We also include the compile times of the backend logic synthesis tool, namely, Synplify Pro 7.1 [13] where the times vary from 2.1 seconds to 872.4 second.

Table 2 shows the experimental results of the AccelFPGA version 1.4 compiler on nine MATLAB benchmarks on a Xilinx Virtex II device XC2V250. Results are given in terms of resources used, and performance obtained as estimated by the Synplify Pro 7.1 tool executed on the RTL Verilog that was output by AccelFPGA. The resource results are reported in terms of LUTS, Multiplexers, embedded multipliers, ROMS and BlockRAMS used. The performance was measured in terms clock frequency of the design as estimated by the internal clock frequency inferred by the Synplify Pro 7.1 tool, and the latency and throughput of the design in terms of clock cycles by using the ModelSim 5.5e RTL simulator.

Table 3 shows similar results for the nine MATLAB benchmark examples on an Altera Stratix EP1S10 device. Resources are measured in LUTS, ATOMS, MACs, and DSP Blocks, and performance is again measured in clock frequency, latency and throughput.

5. Conclusions

This paper described a behavioral synthesis tool called MATCH which reads in high-level descriptions of DSP applications written in MATLAB, and automatically generates synthesizable RTL models and simulation testbenches in VHDL or Verilog. The RTL models can be synthesized using commercial logic synthesis tools and place and route tools onto FPGAs. By linking the two design domains of DSP and FPGA hardware design, MATCH provides DSP design teams a significant reduction in design labor and time, elimination of misinterpretations and costly design rework, automatic verification of the hardware implementation, and the ability of systems engineers and algorithm developers to perform architectural exploration in the early phases of their development cycle. The paper described how powerful directives are used to provide high-level architectural tradeoffs for the DSP designer. Experimental results were reported on a set of nine MATLAB benchmarks that are mapped onto the recent Xilinx Virtex II and Altera Stratix FPGAs.

6. Acknowledgements

This research was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract F30602-98-2-0144.

7. References

[1] Adelante Technologies, A|RT Builder, www.adelantetechnologies.com

[2] Altera, Stratix Datasheet, www.altera.com

[3] Celoxica Corp, Handle C Design Language, www.celoxica.com

[4] System Compiler: Compiling ANSI C/C++ to Synthesis-ready HDL. Whitepaper. C Level Design Incorporated. www.cleveldesign.com

[5] CynApps Suite. Cynthesis Applications for Higher Level Design. www.cynapps.com

[6] G. DeMicheli, Synthesis and Optimization of Digital Circuits, McGraw Hill, 1994

[7] Esterel-C Language (ECL). Cadence website. www.cadence.com

[8] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, "A System for Synthesizing Optimized FPGA Hardware from MATLAB," Proc. International Conference on Computer Aided Design, San Jose, CA, November 2001, See also www.ece.northwestern.edu/cpdc/Match/Match.html.

[9] Mathworks Corp, MATLAB Technical Computing Environment, www.mathworks.com

[10] De Micheli, G. Ku D. Mailhot, F. Truong T. The Olympus Synthesis System for Digital Design. IEEE Design & Test of Computers 1990.

[11] Overview of the Open SystemC Initiative. SystemC website. www.systemc.org

[12] Synopsys Corp, Behavioral Compiler Datasheet, www.synopsys.com

[13] Synplicity. Synplify Pro Datasheet, www.synplicity.com.

[14] Xilinx, Virtex II Datasheet, www.xilinx.com

[15] AccelChip, AccelFPGA Datasheet, www.accelchip.com

Table 1. MATLAB Benchmark Characteristics.

Benchmark	fir16tap	fir64tap	fft64	dec_fir	lms	iirdf1	int_fir	bma	dsl
MATLAB Lines	20	40	98	38	39	33	38	63	175
Directives Used	6	8	9	6	6	6	7	10	9
Verilog Lines	957	1312	4188	1333	2219	883	1084	2758	5654
MATCH Time (sec)	4.0	39.0	10.2	8.9	20.8	2.7	2.5	12.3	38.8
Synplify Time (sec)	3.6	248.7	698.8	32.6	872.4	2.1	9.5	11.9	382.1

Table 2. Results of the AccelFPGA 1.4 compiler on a Xilinx Virtex2 XC2V250 device for nine MATLAB benchmarks.

Benchmark	Resources					Performance		
	LUTS	MUX	MULTs	ROMS	RAMS	Freq (MHz)	Latency (cycles)	Thruput (1/cycle)
fir16tap	373	326	8	8	0	134.2	20	1
fir64tap	1654	330	16	8	16	79.7	59	55
fft64	4212	1473	4	16	2	66.8	5722	4
dec_fir	1356	1209	0	0	0	61.2	8	5
lms	10735	5377	4	0	0	44.2	328	324
iirdf1	119	47	2	0	0	107.1	11	7
int_fir	254	49	1	6	0	75.3	79	75
bma	929	512	0	0	3	72.3	230072	228342
dsl	7145	3055	5	16	0	38.8	3114	2883

Table 3. Results of the AccelFPGA 1.4 compiler on a Altera Stratix EP1S10 device for nine MATLAB benchmarks.

Benchmark	Resources						Performance		
	LUTS	ATOMS	DSP MAC	DSP BLOCK	ROMS	RAMS	Frequency (MHz)	Latency (cycles)	Throughput (1/cycles)
fir16tap	287	547	4	1	1	0	129.8	20	1
fir64tap	2125	2702	16	2	0	16	78.1	59	55
fft64	4439	8361	4	1	0	2	84.3	5722	4
dec_fir	570	1166	1	1	0	0	78.9	8	5
lms	12667	20447	3	3	0	0	48.9	328	324
iirdf1	103	170	3	1	0	0	103.4	11	7
int_fir	311	578	1	1	0	0	67.6	79	75
bma	905	1037	0	0	0	3	57.4	230072	228342
dsl	8514	19905	5	2	0	0	50.3	3114	2883