Building a Multi-Issue Vector DSP with Configurable-Processor Technology

Steven Leibson
Tensilica, Inc.
3255-6 Scott Blvd.
Santa Clara, CA, USA
(408) 327-7335
sleibson@tensilica.com

Beatrice Fu
Tensilica, Inc.
3255-6 Scott Blvd.
Santa Clara, CA, USA
(408) 327-7335
bfu@tensilica.com

The digitization of everything (audio, video, image) creates an insatiable demand for signal-processing horsepower to create, store, transmit, and play back all of this digital content. At the same time, communications and entertainment delivery systems are increasingly portable and use increasing amounts of signal-processing bandwidth, which place tough constraints on the amount of electrical power that can be used to fuel this growing processing load.

DSPs are excellent candidates for performing this digital-content processing because they are programmable and can easily cope with the numerous and changing standards in the whirling world of digital-media processing. However, general-purpose DSPs are not power-efficient for every application due to their general nature.

Hard-wired signal-processing blocks are often more power-efficient but lack the flexibility and programmability of DSPs. Configurable-processor technology bridges the gap between a DSP's fixed-ISA flexibility and programmability and hard-wired power efficiency by enabling the creation of full-featured, programmable DSPs that have precisely the right features for a specific task. Tensilica's Vectra LX—a fixed-point, vector DSP engine created as a configuration option for the Xtensa LX configurable processor—illustrates this concept.

The Vectra LX fixed-point DSP engine is a configuration option for the Xtensa LX microprocessor core. It is a 3-issue SIMD machine with four multiplier/accumulators (quad MAC) and it processes 128-bit vectors as either eight 16-bit or four 32-bit elements. The entire Vectra LX DSP engine was developed in the TIE (Tensilica's Instruction Extension) language and can be modified to suit the target application. As Figure 1 shows, the Vectra LX DSP engine adds 16 vector registers (each 160 bits wide), four 128-bit vector-alignment registers, a second load/store unit, and more than 210 general DSP instructions to the Xtensa LX processor's existing ISA (instruction set architecture).

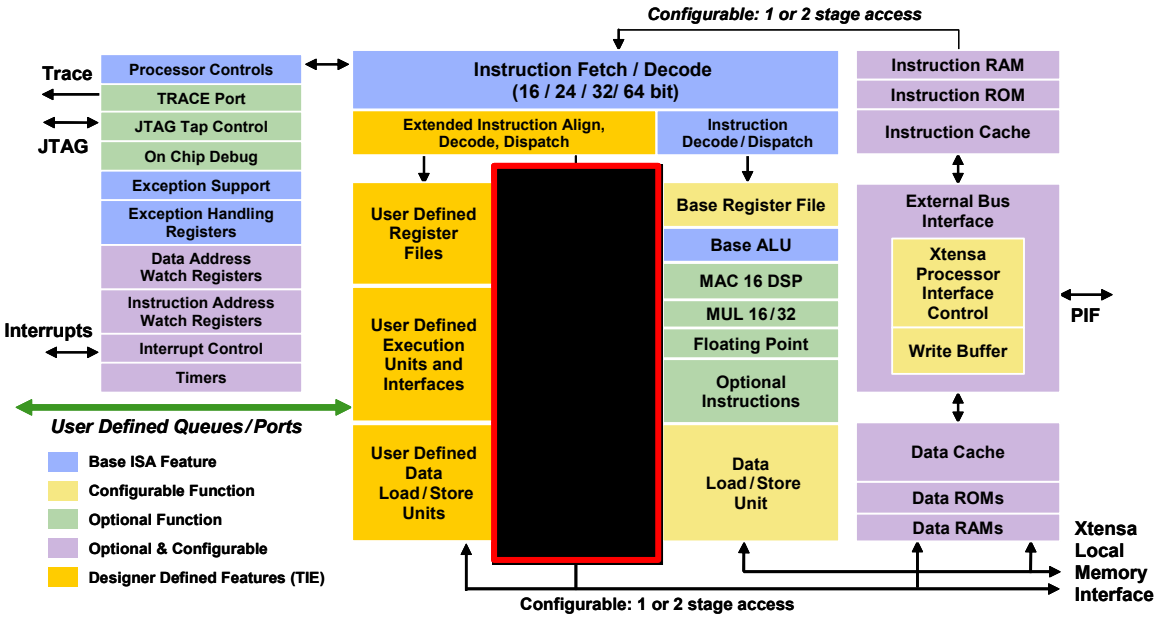

Figure 1: The Vectra LX DSP Engine configuration option adds 16 vector registers (each 160 bits wide), four 128-bit vector-alignment registers, a second load/store unit, and more than 210 general DSP instructions to the Xtensa LX processor's existing ISA (instruction set architecture).

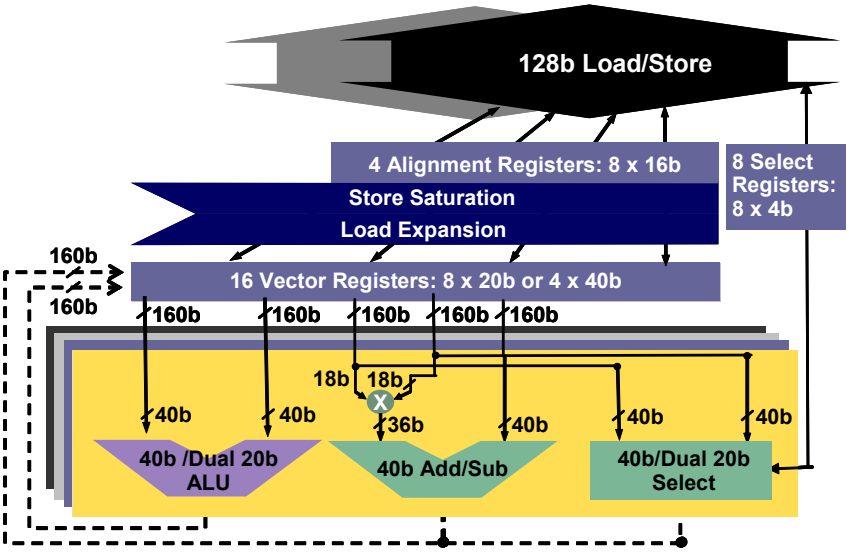

Figure 2: The Vectra LX DSP Engine's 160-bit vector registers feed four identical SIMD units, each consisting of a vector ALU, a separate add/subtract unit, a multiplier, and a select unit. The alignment registers support unaligned loads and stores and serve as intermediate holding registers for load-expansion and store-saturation operations that funnel 128-bit vector data into and out of the 160-bit vector registers.

The base Xtensa LX processor is a single-issue microprocessor with 16- and 24-bit instructions but Tensilica's processor generator offers developers the ability to add wider, multi-operation instructions to the processor's instruction vocabulary through a technology called FLIX (flexible-length instruction extensions). FLIX instructions can be 32- or 64-bits wide and, because the Xtensa LX processor is already designed to handle multiple instruction lengths, multi-operation FLIX instructions can be freely intermixed in the processor's code stream and contiguously packed with existing single-issue Xtensa LX instructions.

When a developer selects the Vectra LX DSP engine configuration option, the RTL for the DSP engine is automatically added to the synthesizable Xtensa LX processor by Tensilica's processor generator. The new Vectra LX instructions are added to the processor's automatically generated software tool set (compiler, assembler, debugger, instruction-set simulator (ISS), and RTOS bindings). The Vectra LX adds 200,000 to 250,000 gates to the Xtensa LX processor's gate count. Most of these additional gates are used to build the Vectra LX DSP engine's registers and execution units because the general-purpose processor and the DSP engine extension share the processor's existing instruction-fetch and –decode units so that hardware need not be replicated although some additional logic is added to decode the new instructions. Figure 2 shows a block diagram of the registers and execution units added by the Vectra LX DSP engine option.

Figure 3 illustrates the 3-operation Vectra LX instruction word format. The four right-most bits of the word are fixed, designating that this instruction is 64 bits wide. The remaining 60 bits of the instruction-word bundle are unevenly trisected into three operation slots: a 24-bit slot and two 18-bit slots. The Vectra LX instruction word's 24-bit operation slot (occupying bits 4 through 27 of the instruction word) accommodates any of the Xtensa LX processor's 80 base instructions including the operations that control the processor's first load/store unit. This operation slot also handles extended, 128-bit load and store instructions for the wide Vectra LX vector registers.

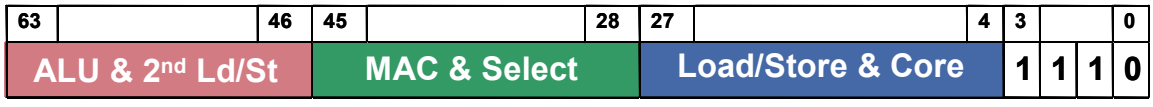

Figure 3: The 64-bit Vectra LX instruction word is unevenly trisected into three operation slots: a 24-bit slot and two 18-bit slots. The four right-most bits of the word are fixed, designating that this instruction is 64 bits wide.

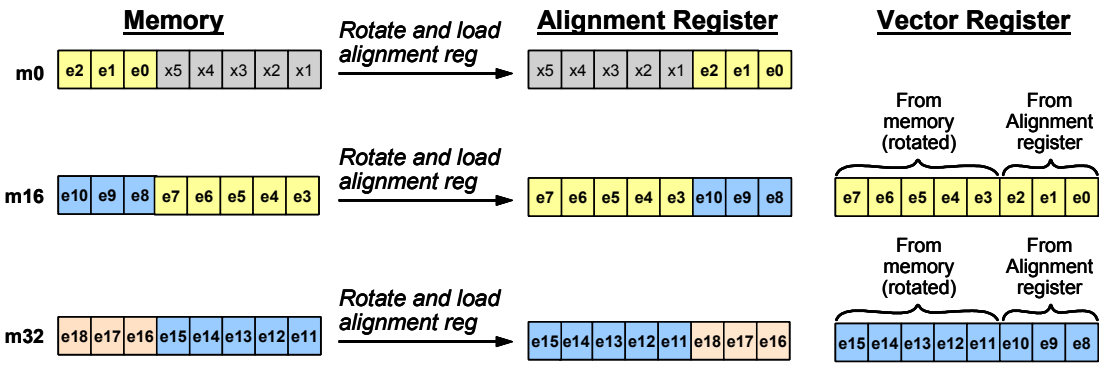

Figure 4: The Vectra LX load/store unit, which is controlled by the 24-bit operation slot in the processor's instruction word, performs both aligned and unaligned load and store operations using its 128-bit alignment registers to hold partial vectors. The processor can perform one unaligned load or store operation per cycle after priming the alignment register with the first partial vector.

The 24-bit-wide operation slot provides enough encoding bits to allow the load and store instructions in this slot to specify aligned- or unaligned-vector loads and stores, as shown in Figure 4. Unaligned loads and stores help a vectorizing compiler deal with memory arrays that have arbitrary data alignment. Compilers can generate code that doesn't always create nicely aligned data arrays which degrades DSP performance. However, this performance loss can be remedied with the right support from the DSP engine.

The Vectra LX DSP engine's alignment registers provide partial-vector storage and are initialized with the first partial vector at the beginning of a string of unaligned loads or stores. Subsequent unaligned load or store operations implicitly merge the new vector data by rotating it and concatenating it with the appropriate portion of the alignment register's contents, creating the full vector. These unaligned load and store operations also prime the alignment register for the next unaligned load or store so that a continuous series of unaligned loads or stores can move data into or out of unaligned data arrays with nearly the same efficiency as that achieved for aligned data.

The DSP engine uses the first 18-bit operation slot (bits 28 through 45) for its 4x40-bit SIMD MAC operations. This operation slot also directs the DSP engine's select operations, which can assemble a register of eight 16-bit vectors drawn from vectors contained in two source-vector registers, as shown in Figure 5.

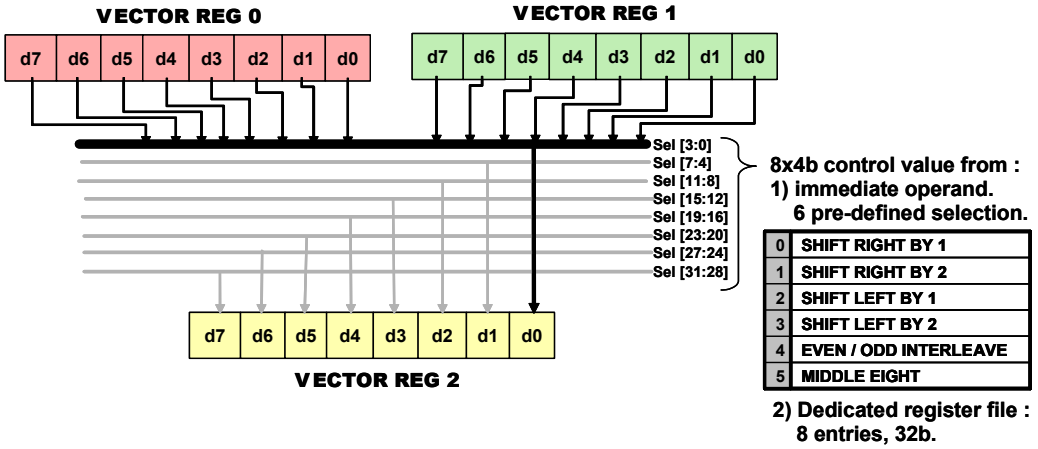

Figure 5: The Vectra LX DSP engine's comprehensive set of select operations can assemble a register of eight 16-bit vectors drawn from individual vectors contained in two other vector registers. The select operations can be used to implement vector operations such as replication, rotation, shifting, and interleaving.

256-Point Radix-4 Complex FFT

ALU	MAC & SELECT	Core & Load/Store
		loopnez a6, 400420c)
add20 v3, v8, v2;	rmulr18 v1, v11, v12;	lvs16.xu v0, a14, a8
pack40 v2,v9, v10;	sel v6, v6, v5, s2;	lvs16.xu v4, a14, a8
nop;	imulr18 v7, v11, v12;	lvs16.xu v5, a14, a8
add20 v8, v0, v5;	nop;	lvs16.xu v9, a14, a8
add20 v2, v4, v9;	rmulr18 v10, v6, v13;	svs16.xu v2, a7, a8
sub20 v11, v8, v2;	nop;	lvs16.i v12, a4, -16
sub20 v9, v4, v9;	rmulr18 v4, v11, v12;	svs16.xu v3, a7, a8
pack40 v2,v7, v1;	sel v1, v9, v9, s0;	svs16.xu v14, a7, a8
sub20 v7, v0, v5;	imulr18 v0, v11, v12;	nop;
sub20 v5, v7, v1;	nop;	lvs16.i v12, a4, -32
sub20 v6, v7, v1;	imulr18 v9, v6, v13;	svs16.xu v3, a7, a8
pack40 v14, v0, v4;	sel v11, v6, v5, s1;	nop;
92%	*100%*	*83%*

Utilization →

Figure 6: This 256-point, radix-4, complex FFT code shows that the Vectra LX DSP engine's four MAC units are fully utilized (as shown in the center column of Figure 6) and the other two instruction slots are heavily utilized.

The Vectra LX select operations can be used to implement vector operations such as replication, rotation, shifting, and interleaving. The second 18-bit operation slot (bits 46 through 63) holds the DSP engine's 4x40-bit and 8x20-bit SIMD ALU operations and operations that control the processor's second load/store unit, which can perform aligned, 128-bit vector loads and stores.

With three operation slots, the Xtensa LX/Vectra LX processor is very efficient at executing DSP code. Figure 6 shows the compact loop for a 256-point, radix-4, complex FFT. The four MAC units are fully utilized (as shown in the center column of Figure 6) and the other two instruction slots are heavily utilized. As a result of this organization, the Vectra LX DSP engine greatly reduces the number of cycles that the Xtensa LX processor uses to perform DSP tasks, as shown by the 256-point, radix-4 FFT numbers listed in Figure 7.

A minimal Xtensa LX processor configuration requires more than 155,000 cycles to compute the 256-point, radix-4, complex FFT. Adding a 32-bit multiplier to the base processor configuration drops that cycle count by nearly an order of magnitude, to 23,633 cycles. Adding the SIMD and multi-operation/cycle abilities of the Vectra LX DSP engine drops the cycle count by more than two more orders of magnitude, to 994 cycles for a performance increase of about 156x.

Exploiting DSP architectural performance requires that associated software tools both understand and effectively use the available architectural features of the DSP. A vectorizing C/C++ compiler called XCC supports the Xtensa LX/Vectra LX architecture. This compiler automatically handles vector data types in the same way that it supports conventional C data types (see Figure 8). Vector type conversions are implicit.

XCC seeks opportunities to vectorize code, primarily within loops, to achieve higher processor performance using the Vectra LX SIMD execution units (see Figure 9). When compiling DSP code, the XCC compiler produces a string of serial Vectra LX operations, schedules these operations, and packages them into wide Vectra LX instruction words (see Figure 10). A scheduling assembler does the same for assembly code.

256pt FFT (Radix-4)

Simple RISC Task Engine	Minimal Configuration Xtensa LX processor using software multiply	155,389 cycles
Scalar Performance	Base Xtensa LX processor with MUL32 option	23,633 cycles
FLIX Performance	Xtensa LX with Vectra LX option	994 cycles

Figure 7: Adding SIMD and multi-operation/cycle abilities to the Vectra LX DSP engine drops the cycle count of a 256-point, radix-4 FFT to 994 cycles, resulting in a performance increase of about 156x.

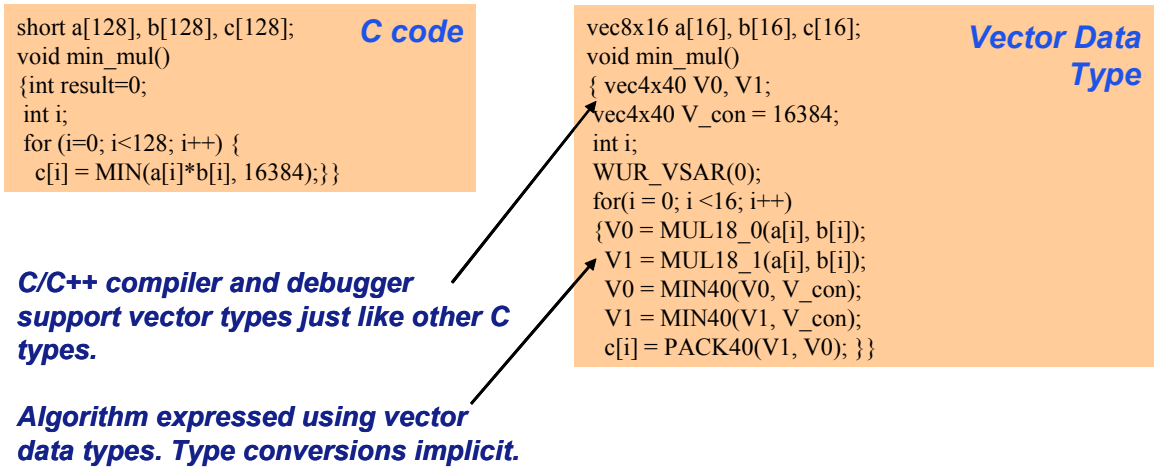

Figure 8: The XCC vectorizing compiler automatically handles vector data types in the same way it supports conventional C data types. Vector type conversions are implicit.

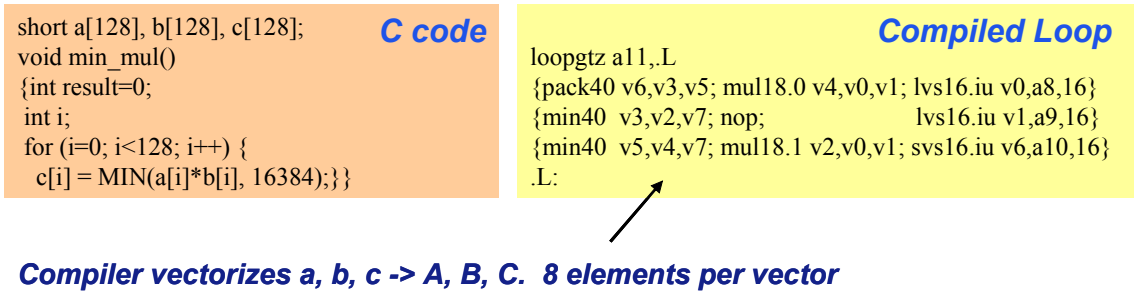

Figure 9: The XCC vectorizing compiler looks for opportunities to vectorize code, primarily within loops, to achieve higher processor performance using the Vectra LX SIMD execution units.

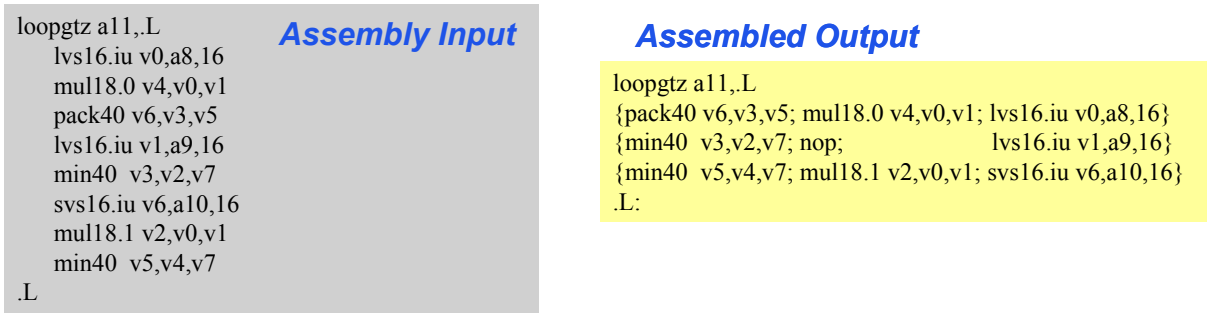

Figure 10. The XCC compiler produces a string of serial Vectra LX operations but it is the scheduling assembler that bundles these operations into the wide Vectra LX instruction words.

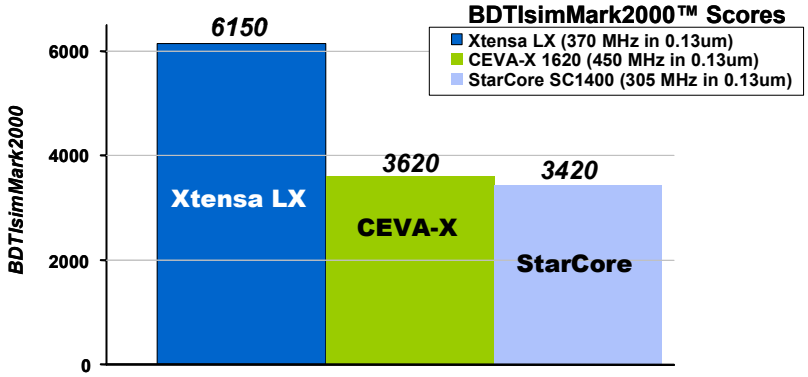

Figure 11: The Xtensa LX/Vectra LX processor augmented with 11 DSP instructions delivers the fastest BDTIsimMARK2000 benchmark score certified by BDTI for a DSP core.

Application performance demonstrates the true worth of new architectural enhancements made to a processor or DSP. You can see an indication of the sort of performance increase the FLIX enhancements make to the Xtensa LX processor's computing abilities from its BDTIsimMARK2000 benchmark score, shown in Figure 11. The BDTIsimMARK2000 is a summary measure of DSP speed distilled from a suite of DSP benchmarks developed and independently verified by Berkeley Design Technology, Inc. Because it is based on realistic DSP algorithm kernel benchmarks, the BDTIsimMARK2000 benchmark characterizes a processor's signal-processing performance far more accurately than traditional simplified measures such as MIPS or MFLOPS.

Figure 11 shows the Xtensa LX/Vectra LX processor's BDTIsimMARK2000 benchmark scores and compares these scores with the scores of two other very fast DSP cores, the CEVA-X and StarCore SC1400 DSPs. The BDTIsimMARK2000 benchmark is a simulation benchmark. For this test, the Xtensa LX/Vectra LX processor was simulated at a clock speed of 370 MHz, which this processor can realistically achieve in 130-nanometer IC process technology using standard synthesis and place-and-route tools.

Note that the Xtensa LX/Vectra LX benchmark score was not achieved solely through the addition of the Vectra LX DSP engine to the Xtensa LX processor. Another 11 custom instruction extensions were developed in the TIE language and refined with BDTi's help to further enhance the processor's performance on this benchmark. Because the Xtensa LX processor is configurable, the extensions created expressly for the BDTi benchmark directly mirror the use of the processor in real applications. In fact, the entire Vectra LX DSP engine was developed in TIE (see Figure 12) and can be modified by the SOC developer to suit the target application. The primary reason for using a configurable processor is to fully exploit the ability to mold that processor's abilities to the target application. In this case, the BDTIsimMARK2000 benchmark is the target application but the target application could just as well have been audio, video, image, or any other type of signal processing.

Define DSP register file 16 entries 160b wide

Define DSP instruction: SIMD Multiply/Add of even elements

Specify pipeline schedule for Multiply/ Add instruction

```
regfile vec 160 16 v

operation MULA18.0 {inout vec acc, in vec m0, in vec m1} {} {
  wire [39:0] sum0 = TIEmac(m0[ 17:  0], m1[ 17:  0], acc[ 40:  0]);
  wire [39:0] sum1 = TIEmac(m0[ 57: 40], m1[ 57: 40], acc[ 79: 40]);
  wire [39:0] sum2 = TIEmac(m0[ 97: 80], m1[ 97: 80], acc[119: 80]);
  wire [39:0] sum3 = TIEmac(m0[137:120], m1[137:120], acc[159:120]);
  assign accum = {sum3, sum2, sum1, sum0}; }

schedule mula {MULA18.0} {
  use m0  4;  use m1  4;  use acc 5;  def acc 5; }
```

Figure 12: The entire Vectra LX DSP engine was developed in the TIE (Tensilica's Instruction Extension) language and can be modified to suit the target application. Here, three lines of TIE define the DSP engine's SIMD multiplication instruction.