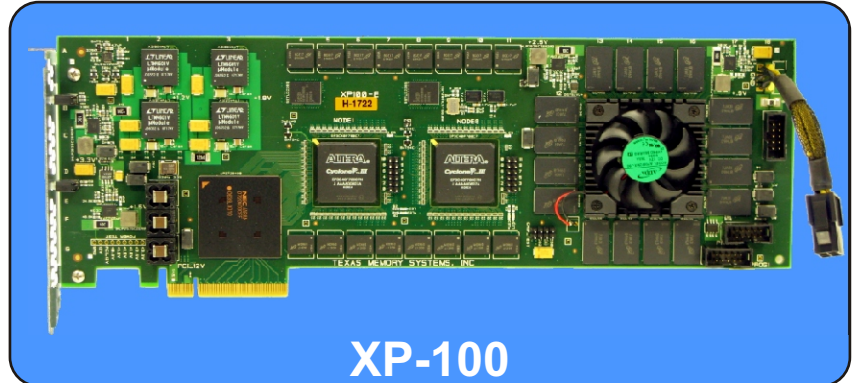


XP-100

100-GFLOPS DSP Accelerator

- 50-GB/s Bandwidth
- Local & Global RAM
- PCIe bus (x8, 1.5-GB/s)
- 500+ Math Library
- Fast Data Acquisition

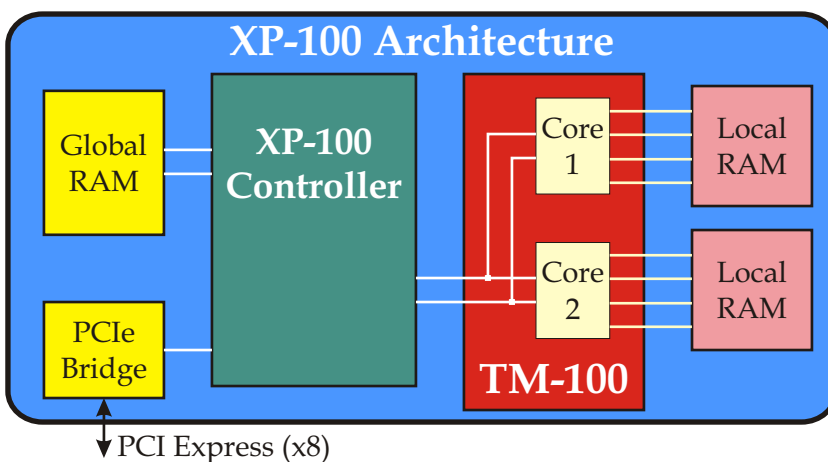


Architecture - The XP-100 is a single board DSP accelerator card that plugs into a PCIe (x8) slot. DSP number crunching power is provided by the TM-100 ASIC chip, designed by Texas Memory Systems, Inc (TMS). The TM-100 has two cores, each with its own local RAM. Its Harvard architecture and VLIW makes the TM-100 a very efficient processor even with 320 floating-point units to control. Also, special hardware units are included for FFT twiddle-generation, and CORDIC Elementary Function Units are included for quick transcendental math.

TM-100 program memory is internal to the TM-100 ASIC, while data is stored in external, tightly coupled local RAM. With a larger local RAM than is usually available on other DSP chips, data can be processed in larger blocks with more complex algorithms. Complementing local RAM is global RAM, which is connected to the XP controller. Global RAM is a staging area for data streams passing between the PCIe bus and the local RAM.

Bandwidth-Bandwidth-Bandwidth - The XP-100 DSP card has extremely fast busses to prevent data starvation at the TM-100 chip. The TM-100 chip has a front-side I/O bus that connects to the XP controller. In addition, the TM-100 chip has a direct and fast connection to its own local RAM. Each core's local RAM can be loaded quickly (5-GB/s) from global RAM while the other core is still running. A continuous stream of data can be received from the PCIe bus (750-MB/s), staged in global RAM, moved into local RAM, processed by the TM-100, returned to global RAM, and then sent to the PCIe bus (750-MB/s) with minimum attention by the host computer.

Math Libraries and Development Environment - The TM-100 DSP chip is software compatible to three generations of TMS DSP processors. The math library consists of over 500 routines (sample routines on page 4) that have evolved over ten years. Likewise, the TMS Program Development System (PDS) has grown to include both the X-Midas and the new QuickXP (Python scripting) development environments. For X-Midas, the PDS includes X-Midas macro language, Fortran, C, and C++ for writing primitives. For QuickXP, the PDS includes Python for writing applications, MacroXP for writing modules in C, and a variety of debugging/performance-measuring tools.



Bandwidth Specifications

Local RAM	40-GB/s	512-MB
Global RAM	5-GB/s	2048-MB
I/O Bus	5-GB/s	
PCIe Bus	1.5-GB/s	

www.superDSP.com

Texas Memory Systems, Inc.



10777 Westheimer, #600
Houston, TX 77042
(713) 266-3200

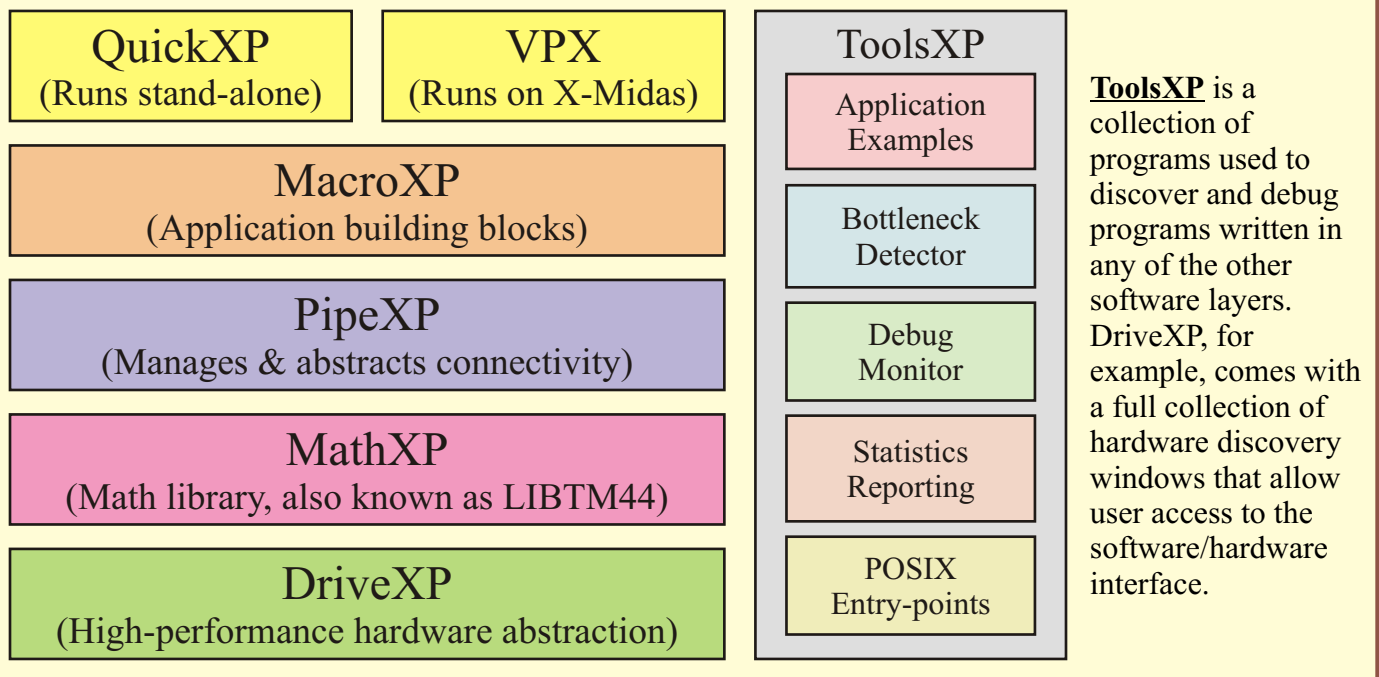
XP-100 Program Development System

QuickXP and VPX sit at the top of the TMS DSP programming toolkit. QuickXP, TMS's stand-alone DSP application environment based on the Python scripting language, allows complex applications to be implemented with minimal lines of code. This allows for extremely rapid application development, yet its script-based solutions are robust and efficient for final user applications. Included extensions provide many useful GUI tools.

The VPX option tree is designed to facilitate the use of TMS hardware in the X-Midas application development environment. It provides a FORTRAN and C interface to the programmer and assists in transferring data between X-Midas pipes and TMS hardware.

Software Layers

TMS software is implemented in layers, with a standardized and transparent API for communication between tiers. This hierarchy results in cleaner, more efficient code, simplifying overall software development and promoting code re-use.



MacroXP is a collection of commonly used DSP algorithms, optimized to execute efficiently on TMS hardware. These macros have been honed into well-rounded, multi-purpose solutions for a variety of large-scale applications. Available macros are: AM/FM/PM demodulators, power spectrum calculator, resampler, sub-band tuner and interference canceller.

MathXP is a time-tested, comprehensive library of 500+ math routines optimized for efficiency on all TMS hardware. Complex data processing algorithms have been boiled down to a simple, consistent interface for operating on large sets of data. MathXP is as easy to use as any other library. The programming paradigm is that library functions are scheduled for asynchronous execution in the DSP accelerator. When subsequent functions are called, they are queued at the hardware. A CPU can issue hundreds of operations for the XP-100. With simple, comprehensive XP-100 control, the host CPU can effectively control 5 to 10 XP-100s with minimal host effort and maximum DSP performance.

QuickXP – Python Scripting Development Environment

QuickXP (Python scripting) is a simple and powerful application development environment for creating massively parallel DSP applications. It provides a scripting interface used to quickly design sophisticated data processing applications, and a C-API for developing optimized, custom data processing components.

QuickXP Features include:

- Python scripting for quick-turn, powerful, multi-process applications
- Maximize computational resources while processing high-bandwidth signals
- MacroXP modules: Resampler, PSC, BOT, AM/FM/PM Demodulators
- Parameters modified real-time while running with immediate results
- Optimized for both throughput and latency
- Optional metadata for controlling down-stream processes byte-by-byte

QuickXP provides two programming abstractions to facilitate the DSP task: pipes and modules. Each DSP task is separated into independent programs called modules that are linked together by pipes. Data flows from module to module through pipes.

The QuickXP Scripting Interface is based on the Python programming language. There is no compile step, so application changes can be tested instantaneously. Maximum performance is maintained as script overhead is kept out of critical sections of code. Complex application programs are reduced to the absolute minimum lines of code. Python is a mature, actively developed language with a thriving developer community. This provides a rich set of third-party tools made instantly available to any QuickXP application.

Example The figure below is an example of a real application, a power spectral calculator, written in minutes using QuickXP. The code is entered almost exactly as it is conceptualized. Creating such real-time data processing applications can be accomplished with four easy steps, each step often translating to a single line of code.

```
import xpq

inpipe = xpq.create()
output = xpq.create()

xpq.modules.rdf("infile.dat",
               inpipe)

xpq.modules.psc(inpipe,
               None, output)

xpq.modules.wrf(output,
               "outfile.dat")
```

1. Pipes will establish dataflow
2. Choose your data source
3. Choose your processing
4. Choose your output method

PipeXP's pipes are optimized for both latency and bandwidth. This is accomplished with out-of-band, data synchronous metadata, with the concept of data flow rate built in. Processing modules know the rate at which data is flowing through the system and adjust processing accordingly, allowing control changes to propagate throughout the entire system within 1 second. This also allows fundamental aspects of the data to change dynamically while the application is running.

Execution Times for 1 Core

FFT (in usec)

Type FFT	Size	Cmpx	Real
Standard	128	0.1	0.1
	256	0.4	0.2
	1K	1.5	1.2
	4K	6.1	4.6
	16K	24.6	18.5
	64K	98.3	73.8
	256K	590.4	393.2
	8M	18900.0	12600.0
Standard, windowed	128	0.2	0.1
	256	0.4	0.3
	1K	2.3	1.5
	4K	9.2	6.1
	16K	36.9	24.6
	64K	147.6	98.3
	256K	786.4	492.8
	8M	25200.0	15800.0
Column	128	0.1	
	1K	1.5	
	4K	6.1	
	64K	147.6	
	512K	1181.0	
Column, windowed	128	0.2	
	1K	2.3	
	4K	9.2	
	64K	196.6	
	512K	1573.0	
2-D	64 x 64	6.1	4.6
	256 x 256	196.6	98.3
	1K x 1K	3146.0	1966.0

Transcendental Math (nanoseconds)

Description	
Complex sine and cosine	2.25*N
Vector arctangent, 1 input	0.75*N
Vector arctangent, 2 inputs	0.75*N
Exponential, selected base	1.5*N
Vector cosine	0.75*N
Vector hyperbolic cosine	3.0*N
Vector division	0.75*N
Vector exponential	0.75*N
Vector logarithm	0.75*N
Logarithm, selected base	1.5*N
Complex vector to polar form	0.75*N
Vector reciprocal	0.75*N
Complex vector to rectangular form	0.75*N
Vector reciprocal square root	1.5*N
Vector sine	0.75*N
Vector hyperbolic sine	3.0*N
Vector square root	0.75*N

Miscellaneous (in nanoseconds)

Description	Complex	Real
Convolution	0.188*NF*N	0.047*NF*N
Convolution, column-oriented		0.188*NF*N
Convolution with dec, Dec=2 or 4	0.188*NF*N	
Convolution with dec, Dec>4	1.5*NF*N	
Matrix multiply	0.188*M*N*P	0.047*M*N*P
Matrix transpose	0.75*M*N	0.375*M*N
Polyphase filter	1.5*N	0.75*N
Inverse polyphase filter	4.132*N	2.066*N
Signal rate resampling		1.5*NF*N
Frequency tuning		0.75*N
AM signal demodulation	0.75*N	
FM signal demodulation	1.5*N	
Auto-spectrum	0.75*N	
Cross-spectrum	1.5*N	
Dot product	1.5*N	0.75*N
Find min/max of vector		0.375*N
Vector add	1.5*N	0.75*N
Vector multiply	1.5*N	0.75*N
Vector-scalar add	0.75*N	0.375*N
Vector-scalar multiply	0.75*N	0.375*N
Vector-scalar multiply and add	1.5*N	0.75*N
Vector add and scalar multiply		0.75*N
Vector mag squared, scalar multiply	0.75*N	
Vector multiply and scalar multiply		1.5*N
Vector-scalar multiply and scalar add		0.375*N
Maximum of two vectors		0.75*N
Minimum of two vectors		0.75*N
Vector average	1.5*N	0.75*N
Vector bitwise logical, 8-bit		0.188*N
Vector bitwise logical, 16-bit		0.375*N
Vector bitwise logical, 32-bit		0.75*N
Vector bitwise logical, 64-bit		1.5*N
Vector bitwise negation, 8-bit		0.094*N
Vector bitwise negation, 16-bit		0.188*N
Vector bitwise negation, 32-bit		0.375*N
Vector bitwise negation, 64-bit		0.75*N
Clip a vector		0.375*N
Square a vector		0.375*N
Cube a vector		0.375*N
Fill a vector with a constant value	0.75*N	0.375*N
Create a ramp vector	0.75*N	0.375*N
Matrix-vector multiply	1.5*M*N	0.75*M*N
Mean of vector elements		0.375*N
Sum of vector elements	0.75*N	0.375*N
Complex weight multiply	1.5*N	
PM signal demodulation	0.75*N	
Vector absolute value	0.75*N	0.375*N
Vector negation		0.375*N
Vector normalization	3.0*N	
Vector compare with bit vector output		0.75*N
Vector threshold	2.25*N	0.375*N
Vector-scalar bitwise logical, 8-bit		0.094*N
Vector-scalar bitwise logical, 16-bit		0.188*N
Vector-scalar bitwise logical, 32-bit		0.375*N
Vector-scalar bitwise logical, 64-bit		0.75*N
Square magnitudes, 50% bin rotation	0.75*N	
Vector avg without scaling	0.75*N*R	0.375*N*R
Find maximum for each vector		0.375*N*R
Pad vectors with specified values	0.75*N	0.375*N
Vector compare		0.75*N
Complex vector from real vectors		1.5*N
Split complex vector, real and imag	0.75*N	