Real-Time Embedded Audio Signal Processing

Paul Beckmann
DSP Concepts, LLC

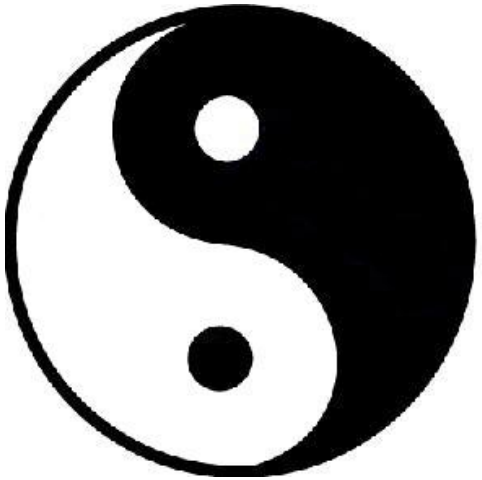
DSP Concepts

Our Vision

Accelerating the development of embedded audio products and technology

Our Mission

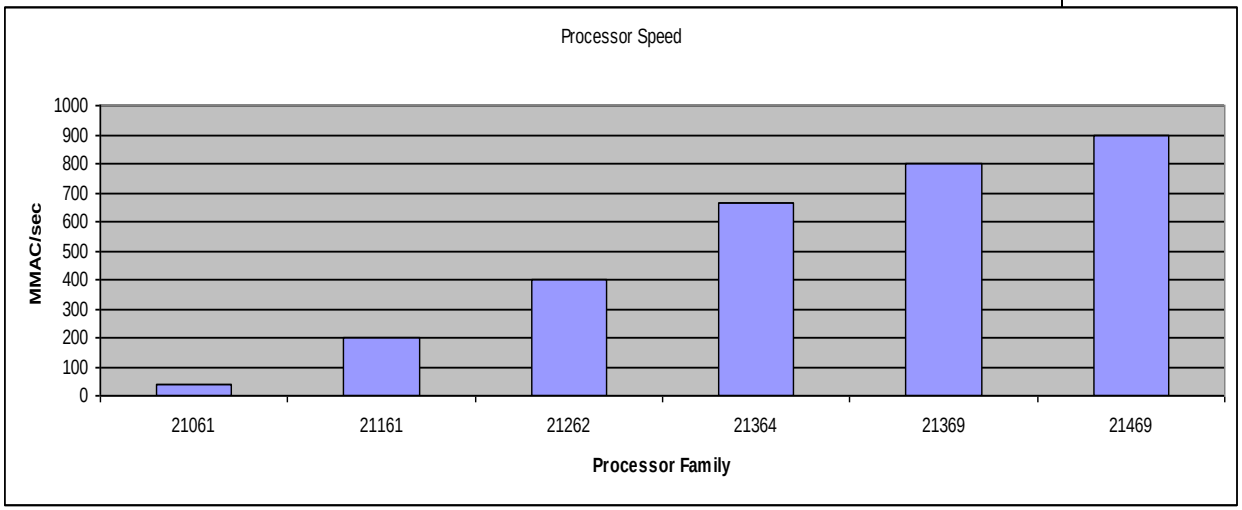
We assist our customers in developing innovative audio products customized to their requirements, through consulting services coupled with state-of-the-art proprietary design tools.

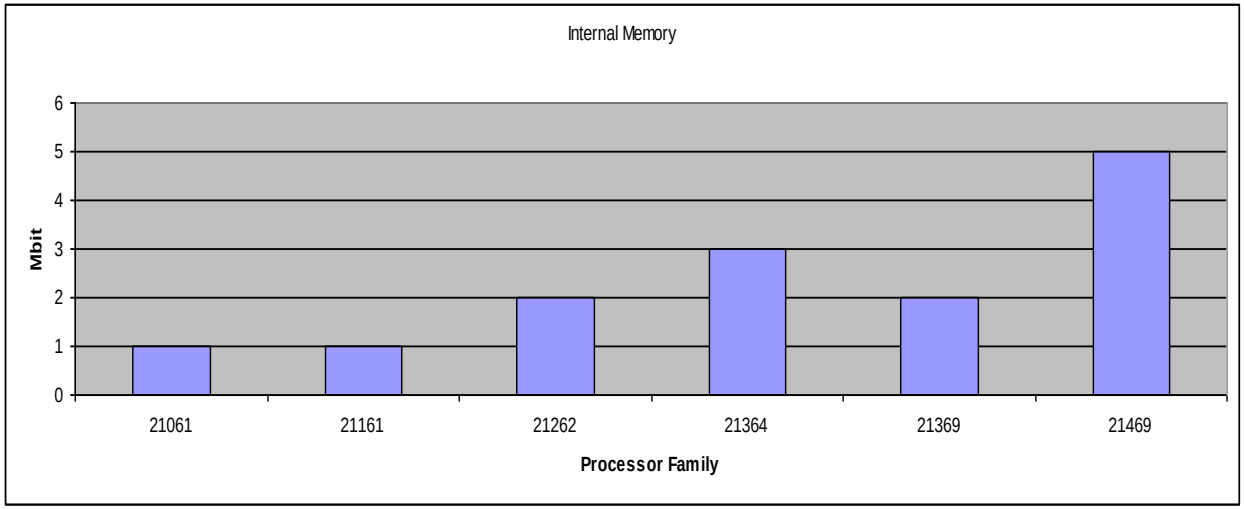


Tools + Services = Value

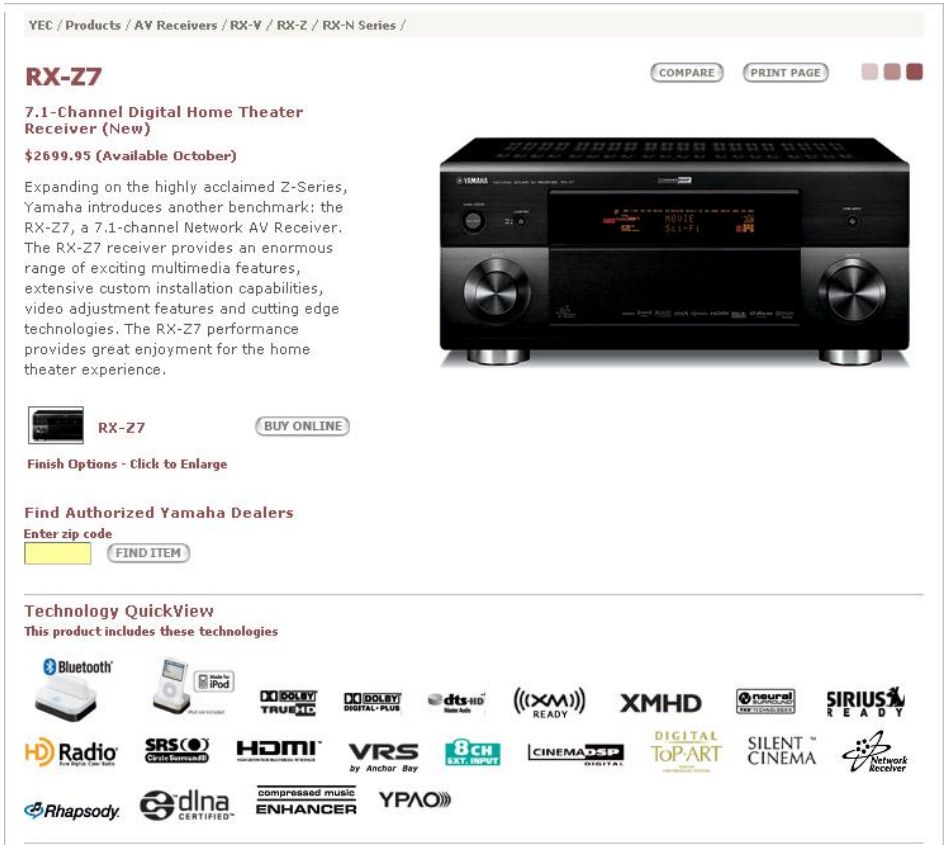
Increasing Capabilities

Successive generations of the Analog Devices SHARC processor family.

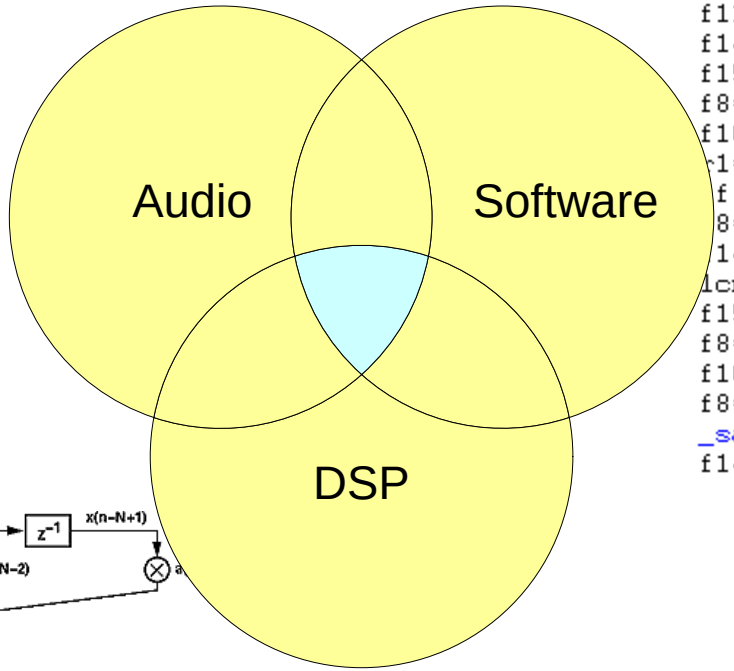




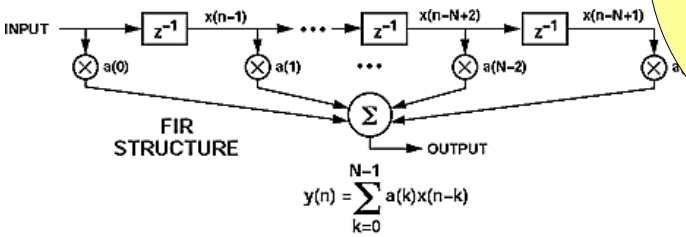
Increasing Complexity

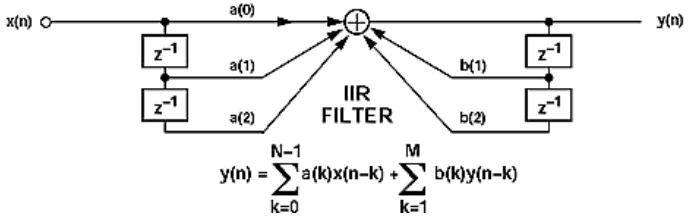


Building Audio Products is Multidisciplinary



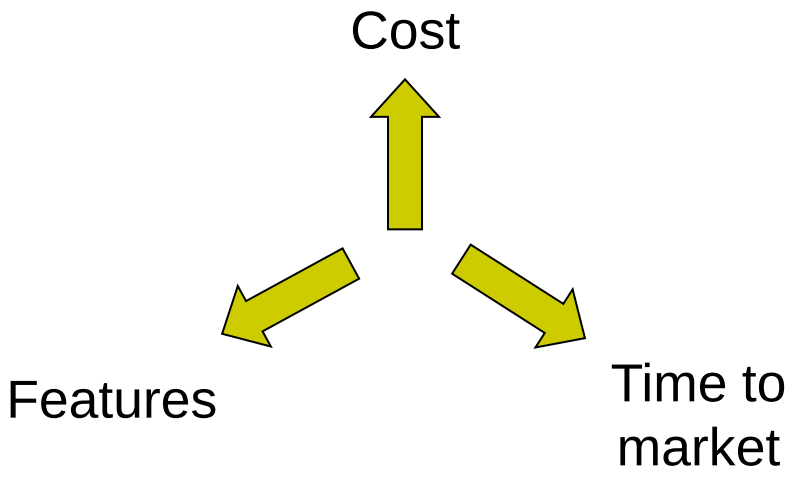

```
r14=0xbf800000;
f5=f5*f14;
f11=f11*f14;
f14=f3*f4;
f15=f0*f11, r8=dm(i4,m4);
f8=f3*f5, f15=f8+f15;
f10=f0*f6, f2=f8+f15, r0=r3;
r1=r1-1;
f le jump _sizeOne_SIMD (db);
8=f2*f7, f15=f10+f14, r3=r2;
14=f3*f4, f12=f8+f15;
lcntr=r1, do _sampleLoopEnd_SIMD until lce
f15=f0*f11, r8=dm(i4,m4);
f8=f3*f5, f15=f8+f15, pm(i12,m12)=r12;
f10=f0*f6, f2=f8+f15, r0=r3;
f8=f2*f7, f15=f10+f14, r3=r2;
_sampleLoopEnd_SIMD
f14=f3*f4, f12=f8+f15;
```





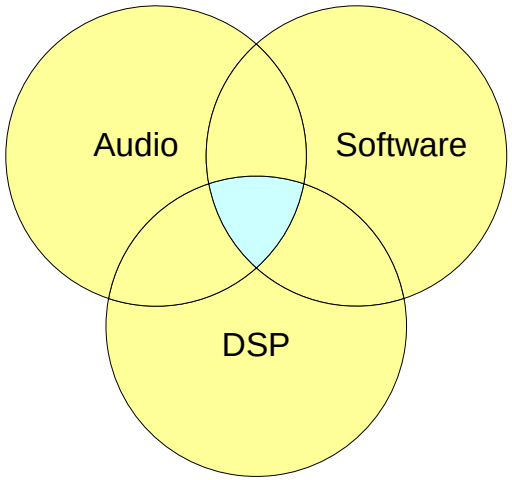
Three Conflicting Development Goals

- You either need to work harder or smarter.
- Audio specific development tools are the solution.



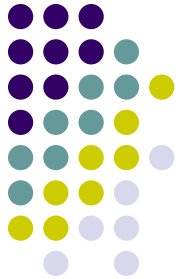
Primary Development Tasks

- Target hardware integration
  - Processor booting and initialization
  - Real-time I/O
  - Host interface
- Develop audio modules
  - Sound quality issues
  - Numerical issues
- Design the audio signal chain
  - Link together audio modules. Standard and custom.
  - Achieve desired sonic effect
- Tune the system
  - Optimize overall sound quality
  - Make changes in real-time
- MIPs and memory optimization
- Testing and Validation



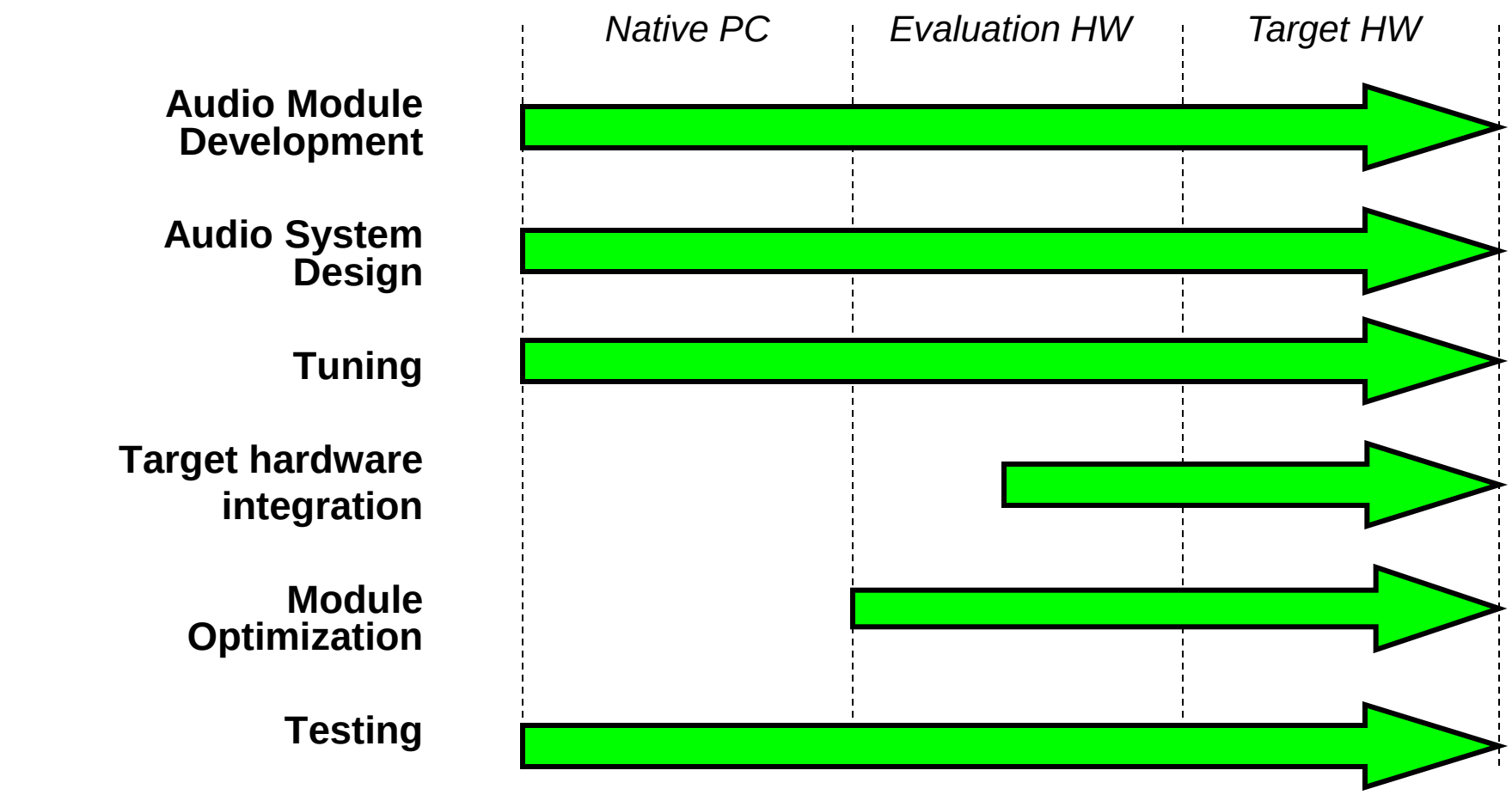
Introducing Audio Weaver

"MATLAB Toolbox for Real-Time Signal Processing"



- Cross platform approach speeds development
  - Prototype on the PC
  - Seamlessly switch to your evaluation hardware and then to your target hardware
- Supports SHARC and Blackfin processors from Analog Devices
- Generates efficient code
  - Links together a sequence of hand optimized processing functions
  - Can be included in commercial products without further MIPs or memory optimization
- Full hierarchical design
- Design systems using MATLAB scripts
- Real-time tuning
- Dynamic memory allocation
- Includes a large library of optimized audio processing modules
- Full regression testing capabilities
- Built-in MIPs and memory profiling

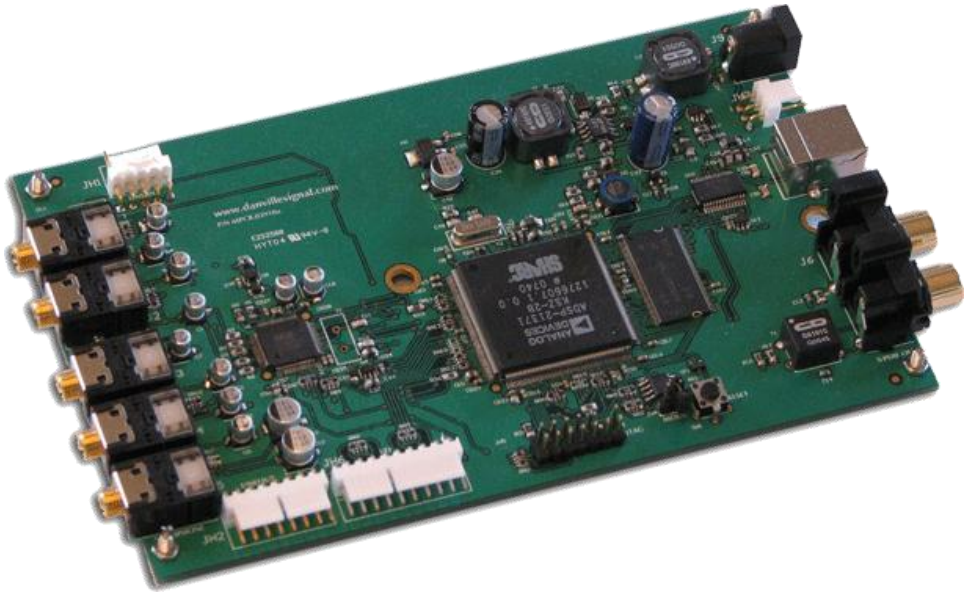
Cross Platform Approach Speeds Development



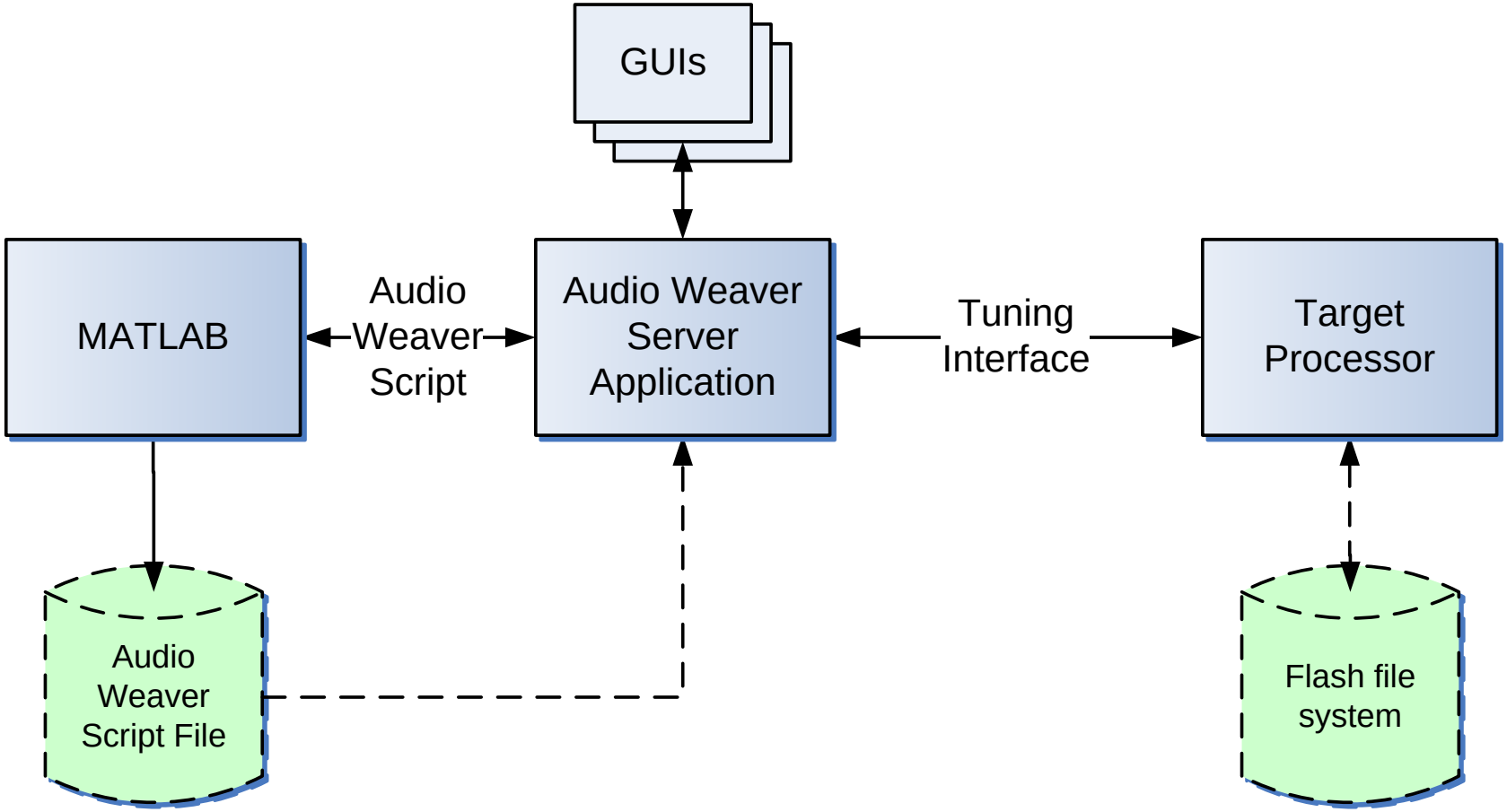
Audio Weaver Demonstration

Audio Weaver Demo Board

- Developed in conjunction with Danville Signal
- SHARC 21371 processor
- USB tuning interface
- USB bus powered
- 144 audio module classes
- 2 in / 4 out analog
- S/PDIF In and Out
- External SDRAM
- SPI flash

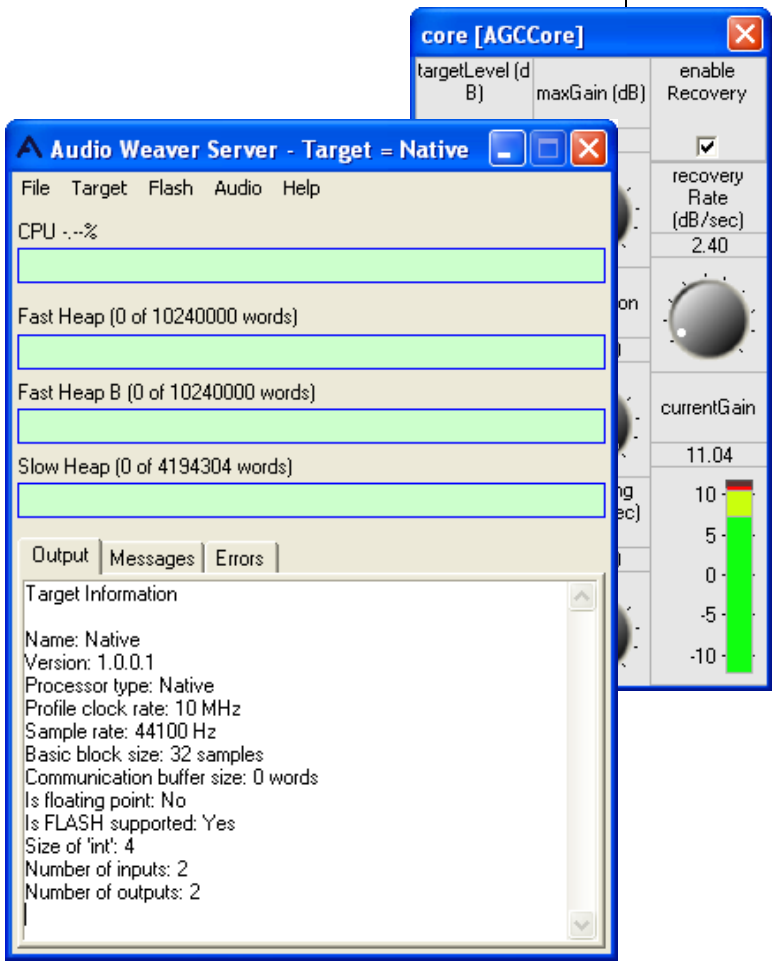


High-Level Architecture

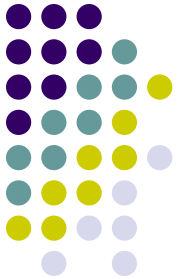


Audio Weaver Server

- PC application that manages the tuning interface to the target processor
- Includes native real-time audio support
- Draws user interfaces
- Translates symbolic names to physical addresses on the target



Built Upon MATLAB

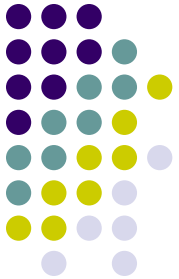


- MATLAB is the standard environment for algorithm development.
- Allows you to leverage MATLAB's huge library of signal processing design functions.
- Provides a full scripting language for system design, parameter setting, tuning, and general automation.
- Scripts lead to consistent and repeatable results.

Native PC Target

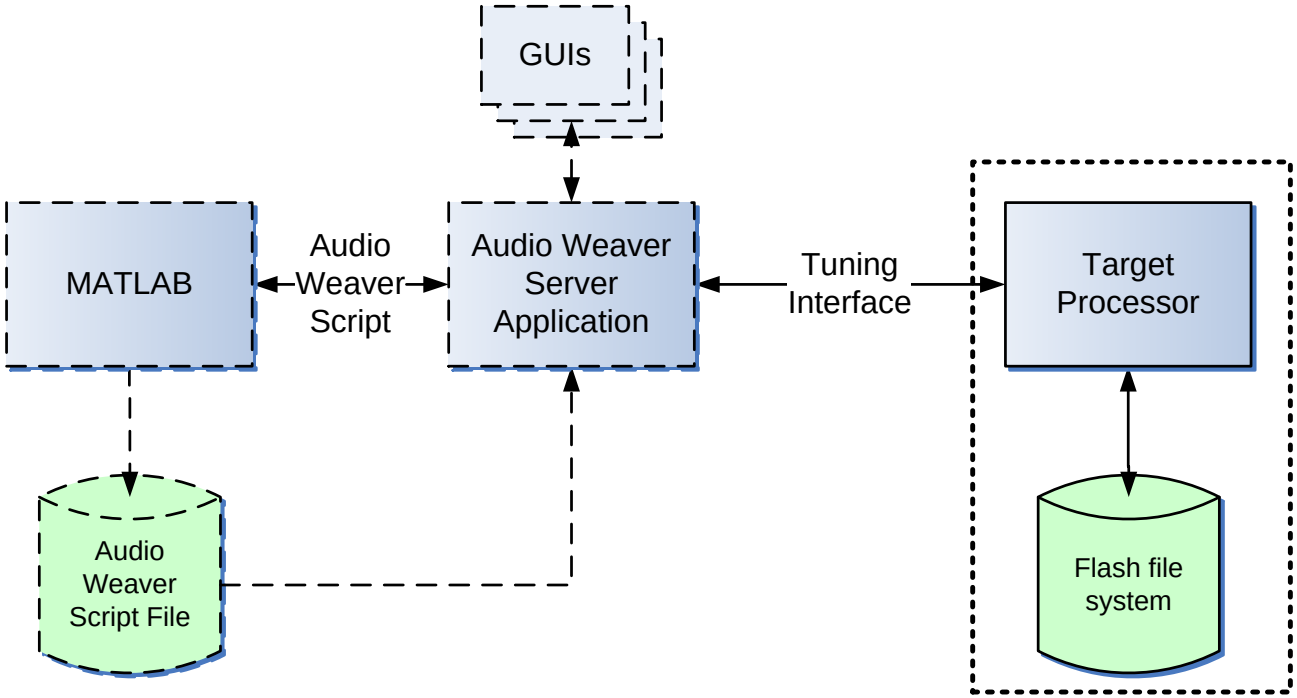
- No additional HW required!
- Supports multichannel USB and Firewire devices
- Many vendors to choose from: Motu, Yamaha, M-Audio, etc.
- Audio input can also be taken from a sound file. WAV, MP3, and WMA, formats supported.
- Full audio module library support.

Flexible Usage Models



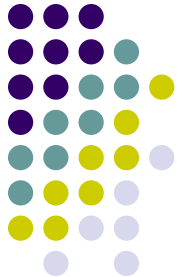
- Development – using MATLAB
- Demonstration - PC Script Execution
- Deployment in an end product

Deployment – Dynamic Allocation



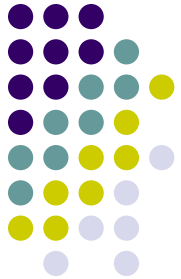
- Audio Weaver commands are stored in the flash file system.
- Commands are executed when the DSP boots.
- Server connects for tuning or to change the audio design.
- No need to rebuild the executable.

Dynamic Allocation Details



- Memory allocated from 3 heaps – matches the SHARC memory architecture.
- Allocation occurs only before audio processing is started.
- Fragmentation avoided by not supporting free(). Must “destroy” and start again.
- Benefits
  - Rapid prototyping, testing, and validation.
  - Update products simply by changing a data file.
  - Provide multiple different product configurations using multiple data files.
  - Segregates platform development from the audio processing development. Matches the automotive amplifier work flow.
  - Reduces the number of DSP development tools licenses.

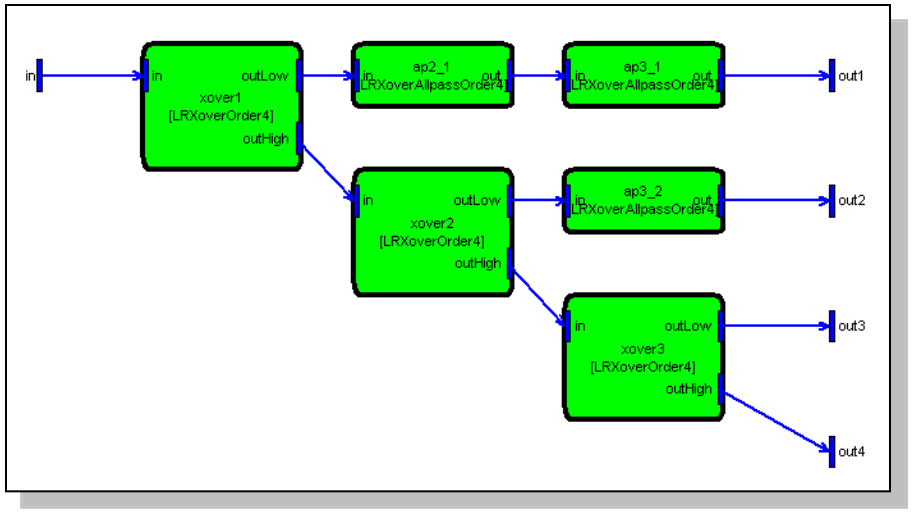
Extensive Audio Module Library

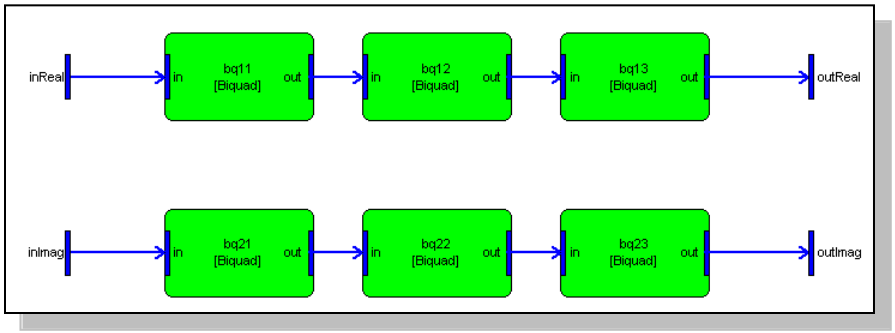


- Basic
  - Scalers
  - Mixers
  - Adders
  - Routers
- Delays
  - Basic delays
  - N-tap delays
- Dynamic
  - Compressors
  - Limiters
  - Automatic Gain Controls
  - Clipping
- Filters
  - Second order filter designer with over 20 filter types
  - Butterworth and Linkwitz Riley crossovers
  - 5 coefficient Biquad filters
  - 32-bit and 40-bit processing
  - FIR filters
- Spatial
  - Balance and fader controls
  - Allpass filters
- Misc
  - Table lookup
  - Meters
  - Signal generators
- And many others!

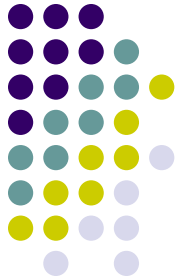
Subsystems Provide Additional Audio Functions

- Crossover filters
  - Butterworth and Linkwitz Riley designs
  - Arbitrary number of output bands
- Hilbert network (90 degree phase shift)
- Graphic equalizer
- Automatic gain control
- Dynamics processors





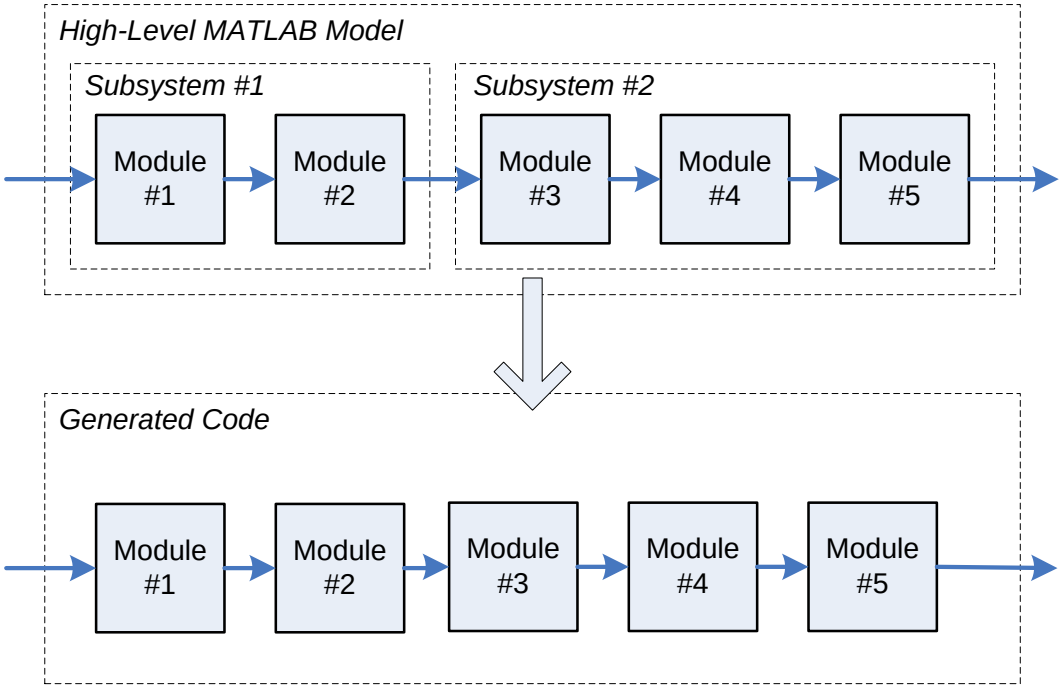
Hierarchy



- “Subsystem” = Modules + I/O Pins + Connections
- Arbitrary nesting of subsystems.
- Hierarchy extends to both the processing and the control functions.
- Two different types of hierarchy
  - “Virtual Hierarchy”
  - “True Hierarchy”

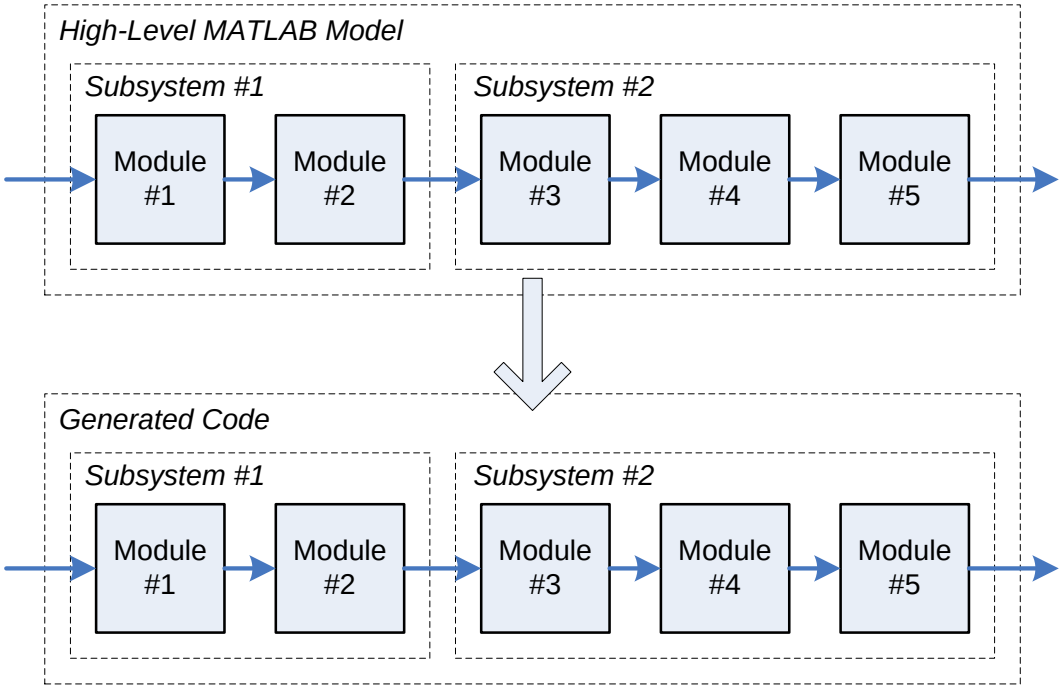
Virtual Hierarchy

- Hierarchy only exists in the MATLAB model.
- Hierarchy is flattened when the system is built. Still appears as hierarchy when viewed from MATLAB.
- Supports dynamic memory allocation.

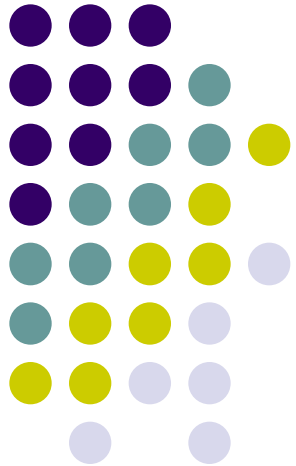


True Hierarchy

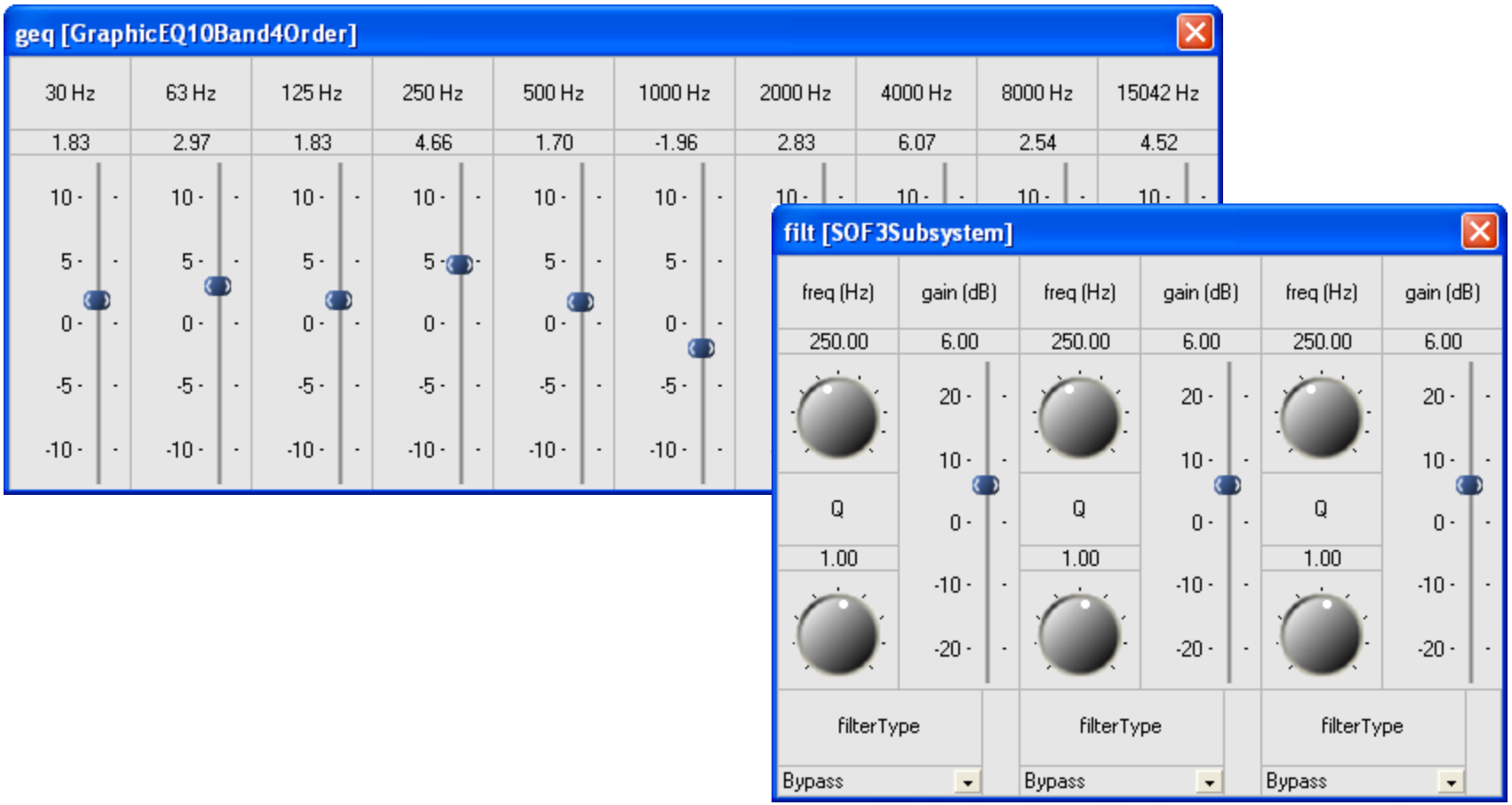
- Hierarchy is maintained in the generated C code
- Creates new audio module classes.
- Requires rebuilding the target executable.
- Used by IP developers



Examples

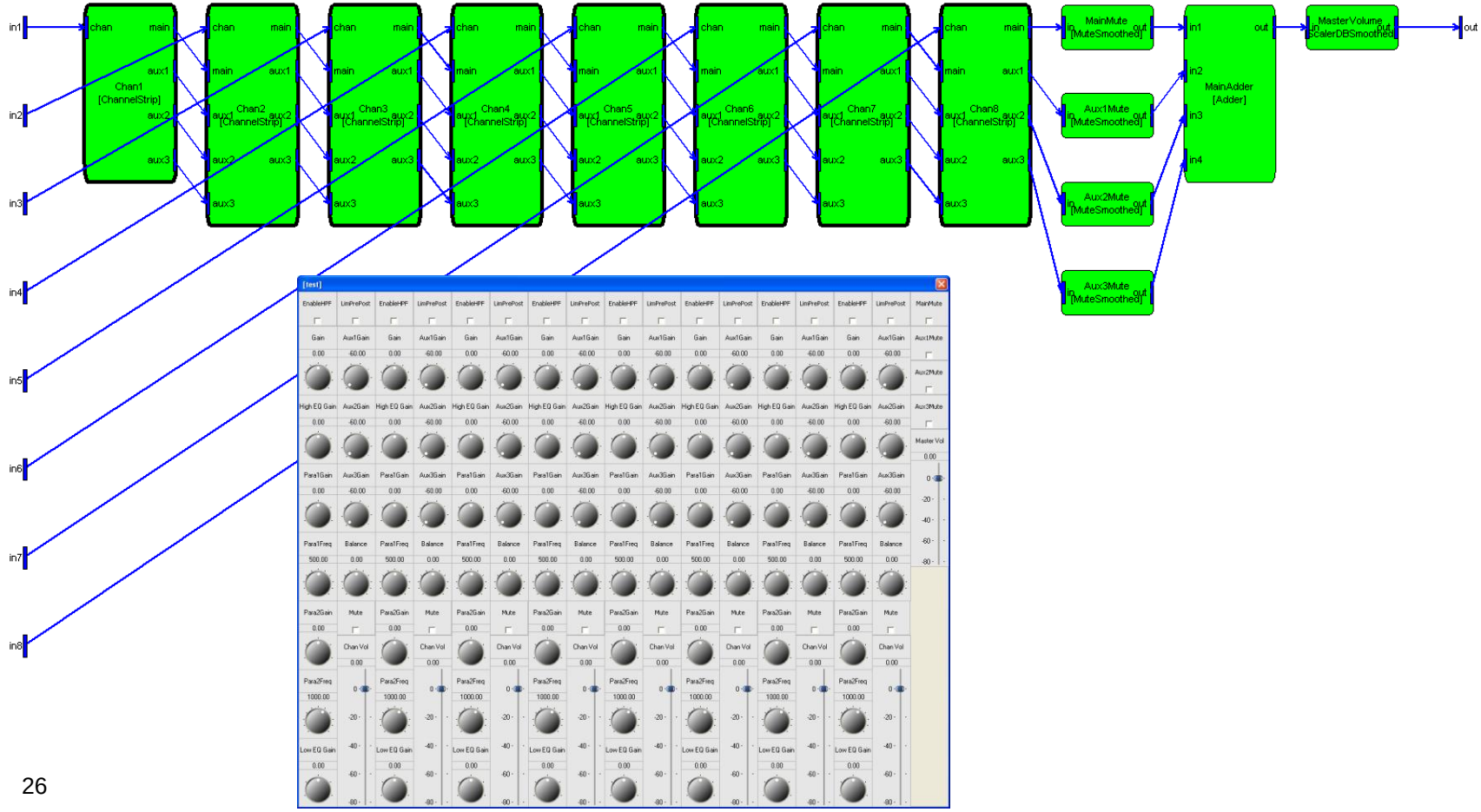


Equalizers
Graphic and Parametric

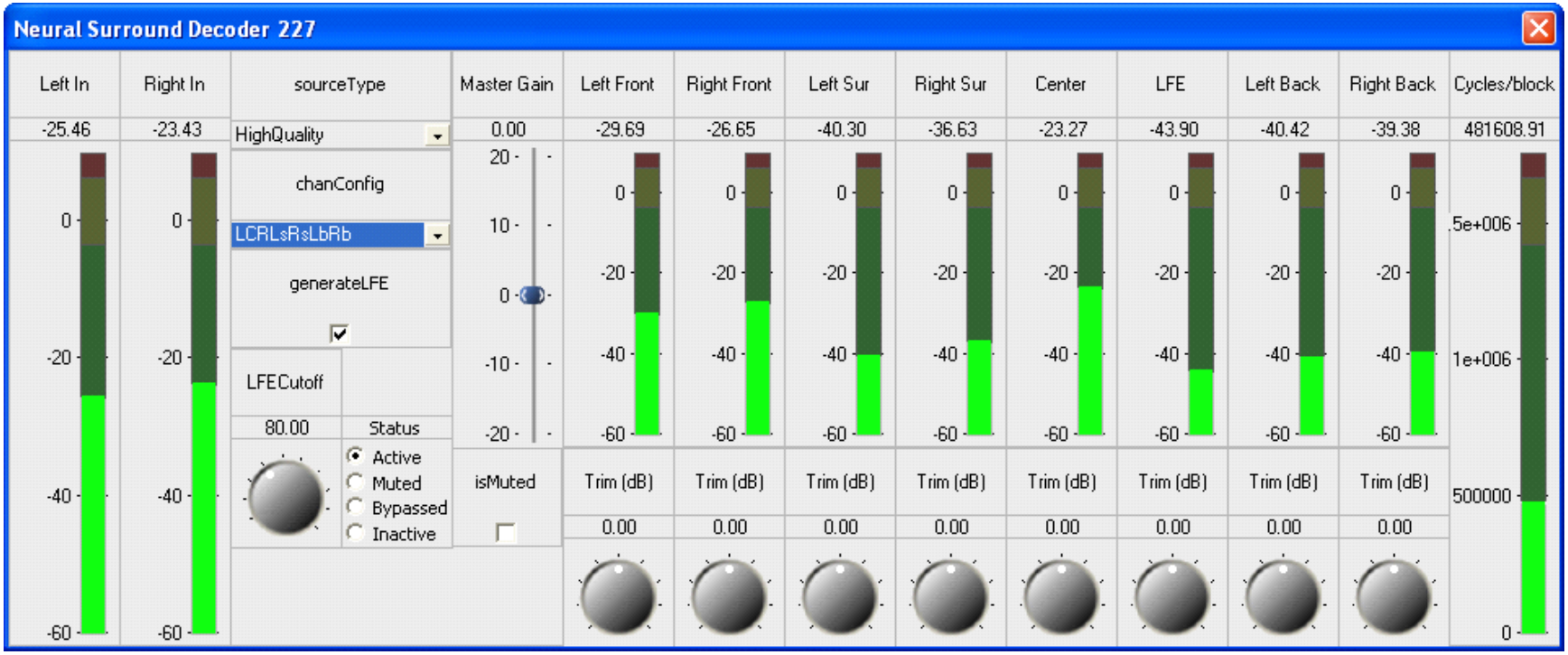


Mixing Console Example

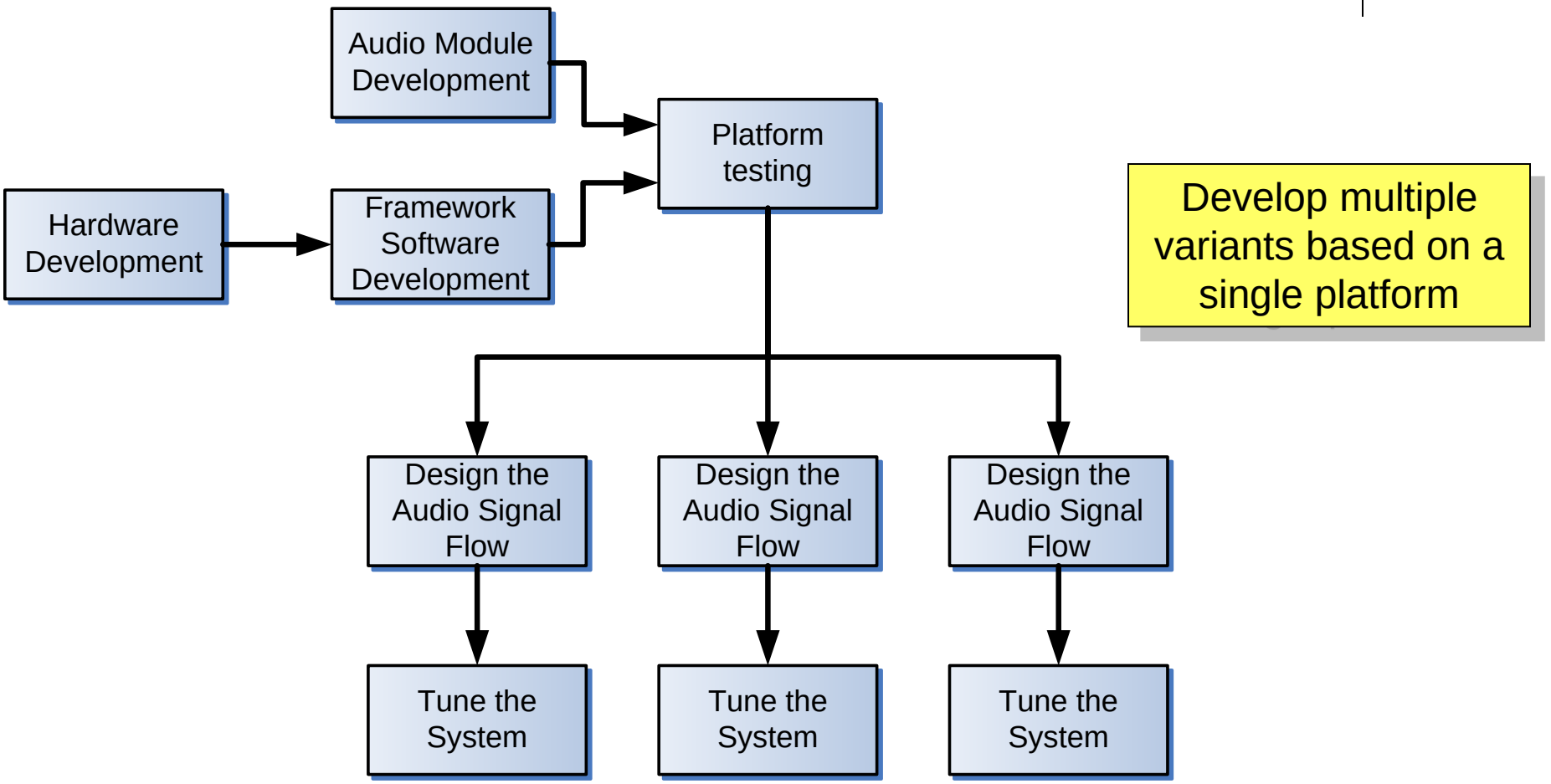
.mixingconsole [Mixing Console]



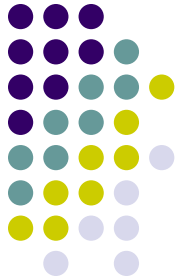
Neural Surround Decoder



Automotive Audio Development

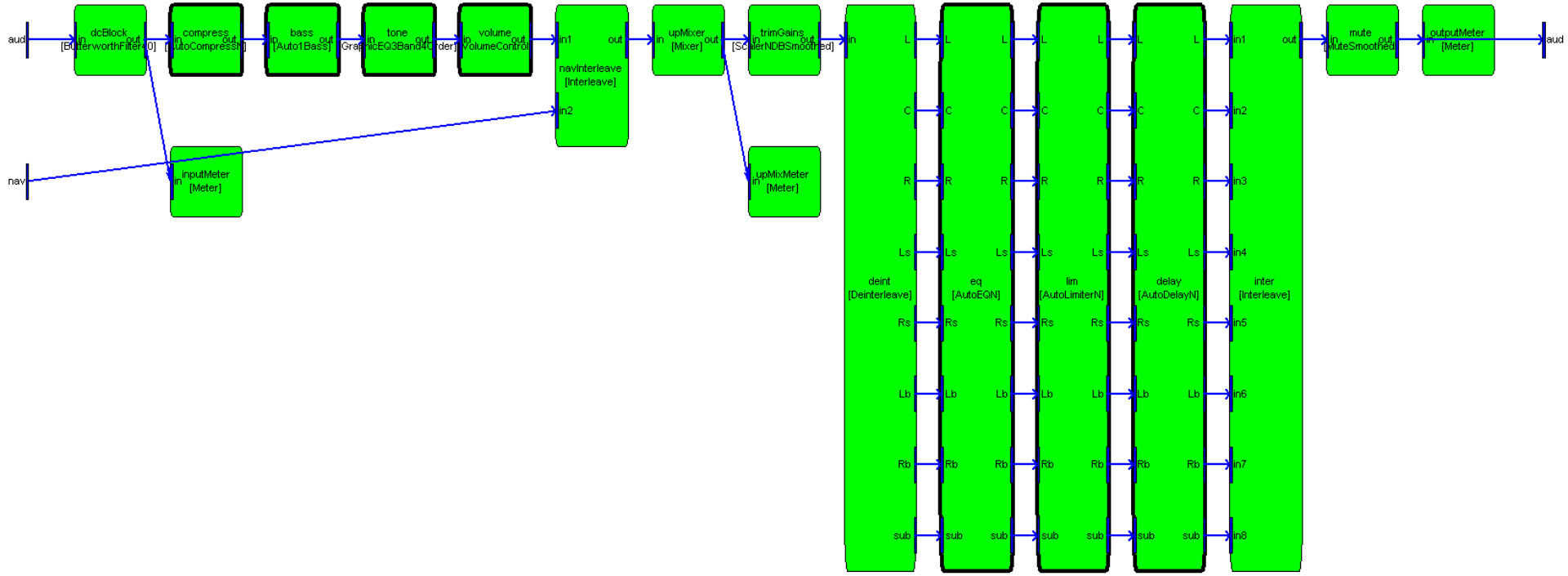


Mid-End Automotive Amplifier

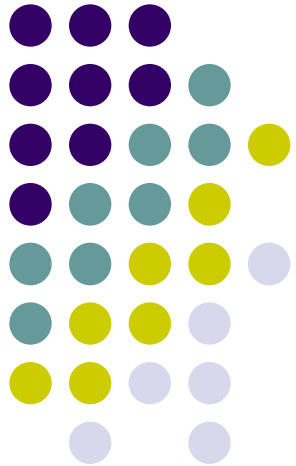


- 6 inputs + Navigation → 8 outputs
- Input Processing includes
  - Highpass DC blocking filter
  - Compressor
  - Bass processing
  - 3 band tone controls
  - Volume control with built in loudness compensation
- Upmixing to 8 output channels
- Output processing (8 channels)
  - 8 programmable EQ sections
  - Limiter + Delay + Mute + Metering
- Memory used by audio processing (buffers & structures only)
  - Fast heap 7997 words
  - Fast heap B 2174 words
  - Slow heap 0
- 32 sample block size
- 54.5% of a 263 MHz SHARC 21371 = 144 MIPs

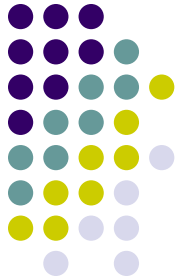
Example (continued)
Automotive Audio Processing



Technical Details

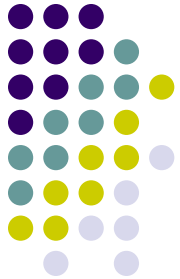


Reusable and Maintainable Audio Processing Code



- Write in C or C++. Avoid assembly.
  - Use block processing
- Optimize only as much as is necessary
  - As little assembly as possible.
  - Separate assembly code into a set of optimized processing functions
- Segregate audio processing and framework code
  - Audio processing is reusable
  - Framework code is target specific
- Develop as much of the audio processing on the PC as possible
  - Superior development environment
  - Unlimited MIPs and memory
  - Forces code to not be processor specific
- Automated testing

Uses Block Processing



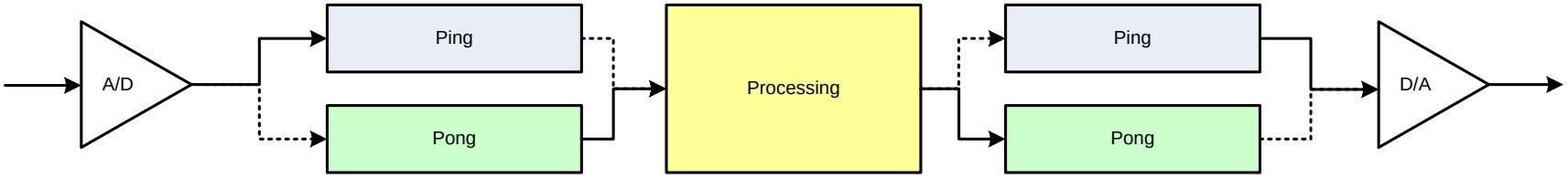
Stream Processing

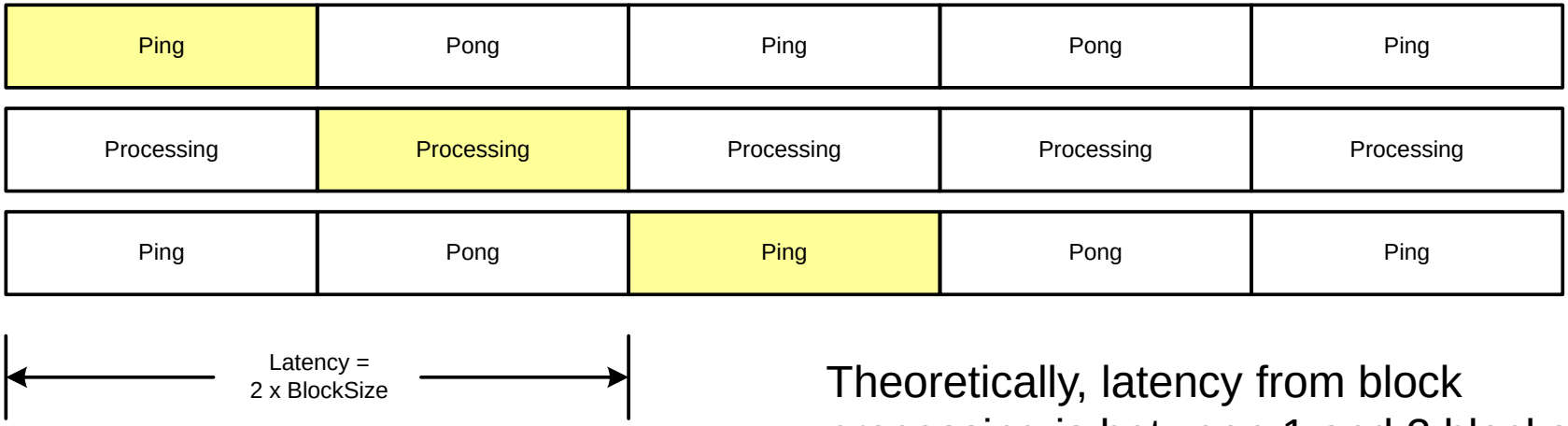
- Process one sample at a time
- Low latency
- Low memory requirements
- Required for reverbs and some recursive algorithms
- Must program in assembly for efficiency
- Difficult to reuse code

Block Processing

- Process blocks of samples at a time
- Use chained DMA to reduce I/O overhead
- Higher latency. Higher memory requirements
- Required for decoders, encoders, and frequency domain processing
- Can program most of it in C
- Leads to reusable code

Block Processing Latency



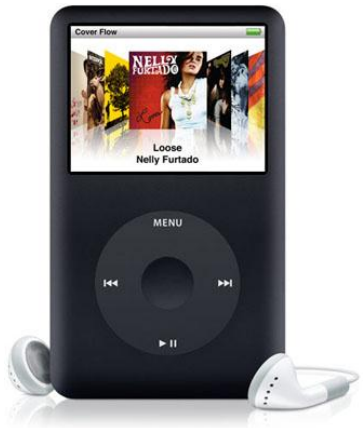


Theoretically, latency from block processing is between 1 and 2 blocks of data. In practice, it is always twice the block size.

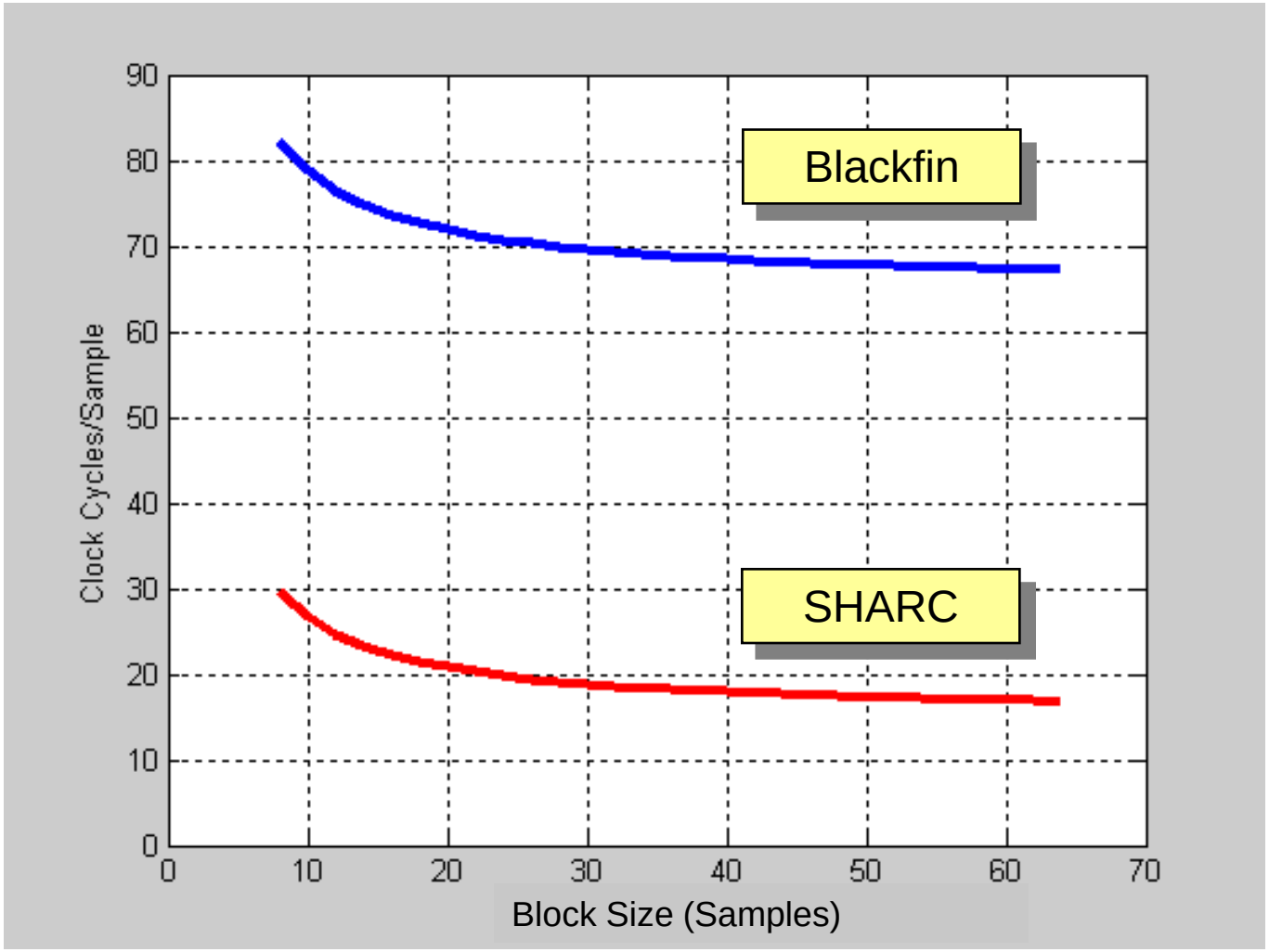
Allowable Latency Varies by Application

- Active noise control (< 2 msec)
- Musical effects (< 5 msec)
- Musical instruments (< 10 msec)
- Live sound / Sound reinforcement (<20 msec)
- Television / AVR (< 30 msec)
- Automotive audio (5 to 30 msec)
- Music players / streaming players (<100 msec)

Latency adds up if there are multiple devices in series.



Block Processing and Efficiency

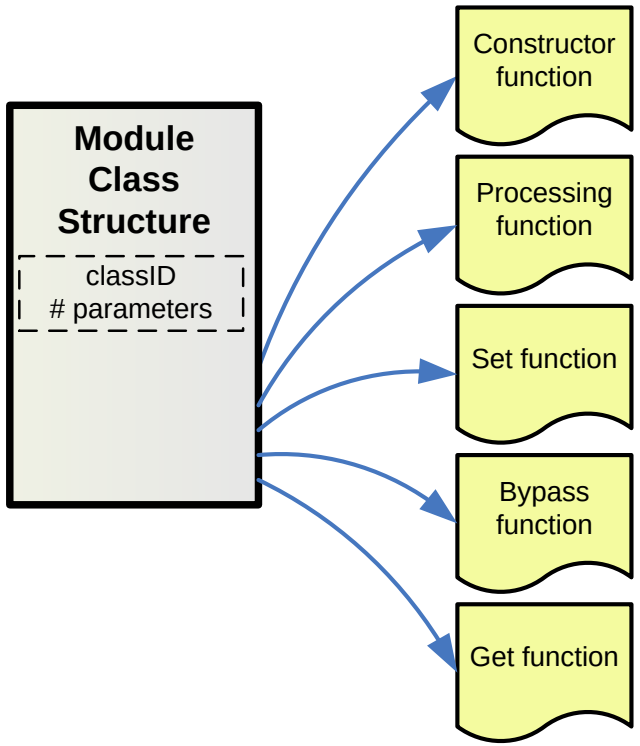


Operations per sample for a 10th order IIR filter. Blackfin implementation is double precision.

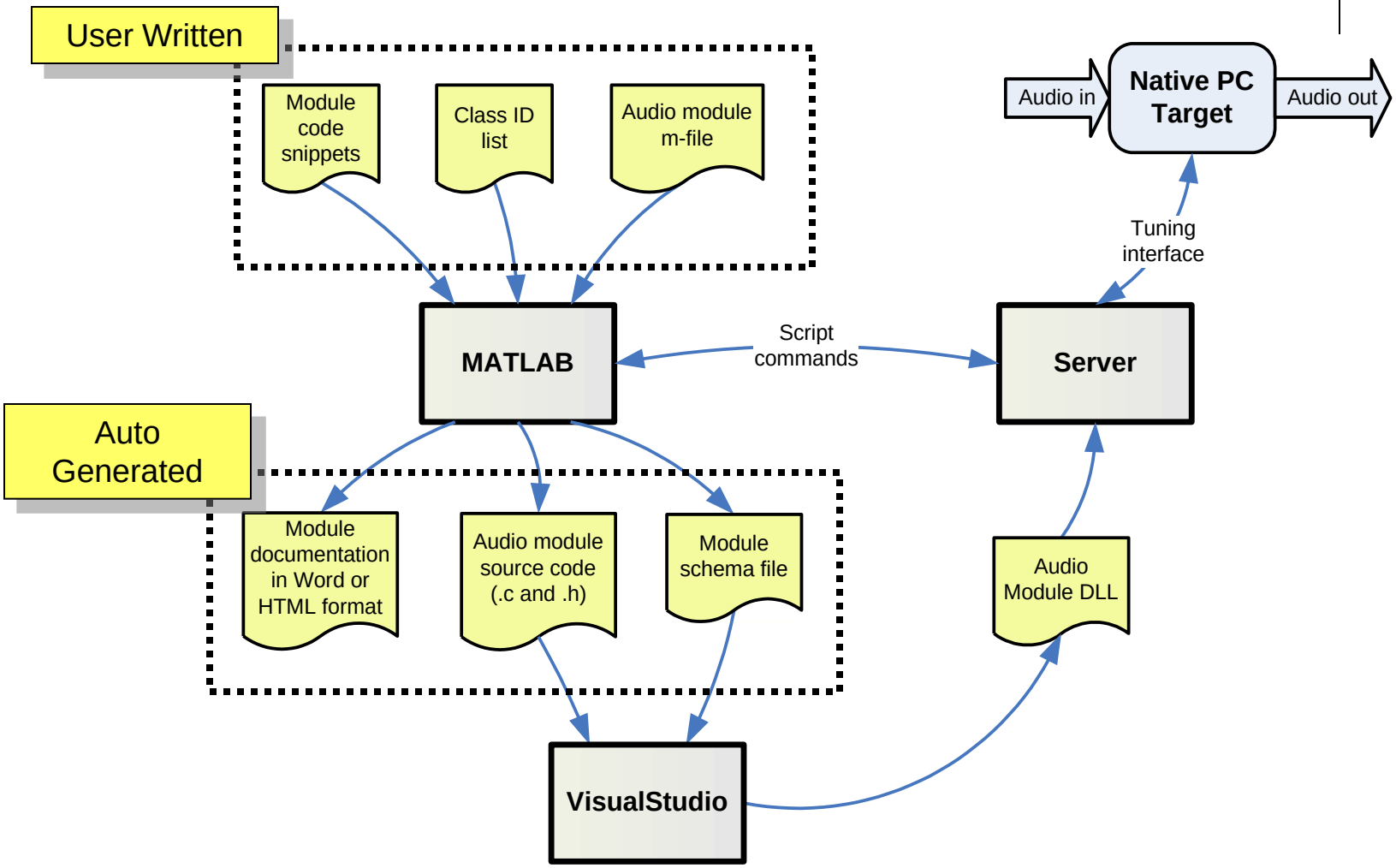
Audio Module Functions

- Constructor function (optional)
- Processing function
- Bypass function (optional)
- Set function (optional)
- Get function (optional)

Everything that is needed to instantiate, run, and control the module

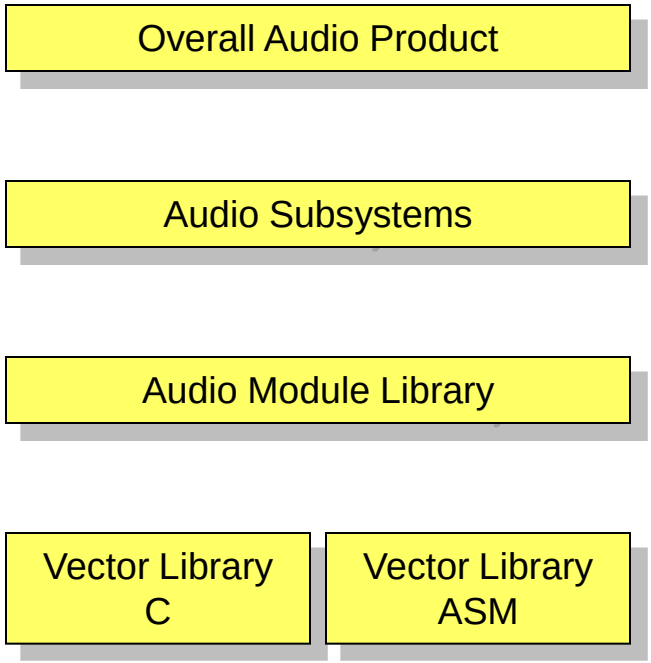


MATLAB Module Generation

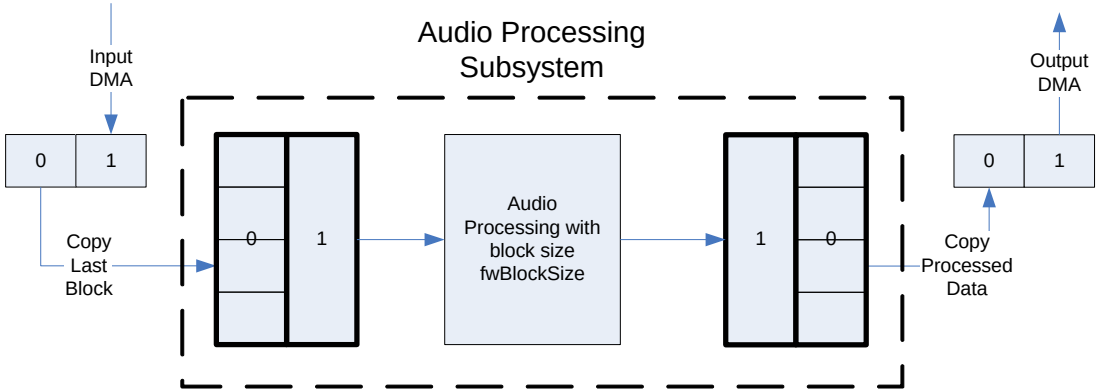


Vector Libraries

- Audio modules are built upon a set of optimized vector functions.
- Each function operates on a block of data
- Simple signal processing primitives
  - Copier
  - Scaler
  - Smoothly varying scaler
  - Biquad
  - Adder
  - Etc
- Each primitive is used by several audio modules. For example, the smoothly varying scaler
  - ScalerSmoothed
  - ScalerNSmoothed
  - MuxSmoothed
  - NoiseGate
  - MixerSmoothed
  - ScalerDBSmoothed
  - MuteSmoothed
  - Balance
  - Router
- C version of each function exists. Used on the PC
- Functions are optimized in assembly for different DSPs
- Other processor families can be supported by simply porting the vector library



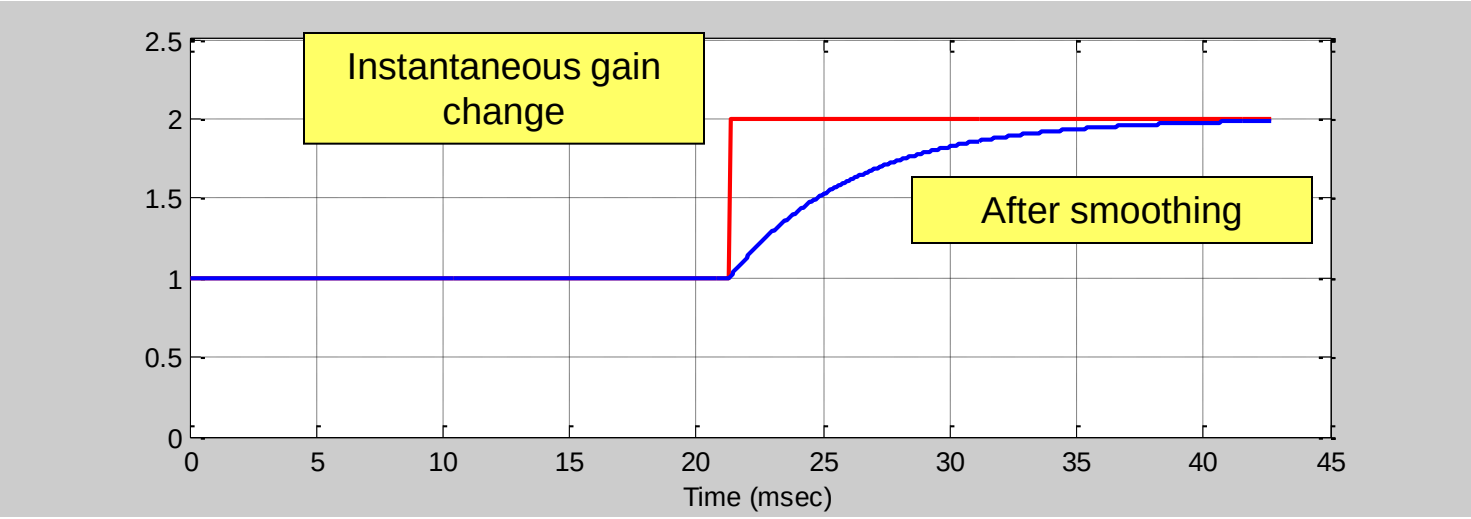
Audio Processing API



- Audio processing segregated from the real-time platform
- All interrupts and threading handled by the platform
- Platform passes blocks of audio into the run-time audio processing
- Audio processing indicates when it has enough data to process
- Platform calls audio processing from a lower priority user interrupt

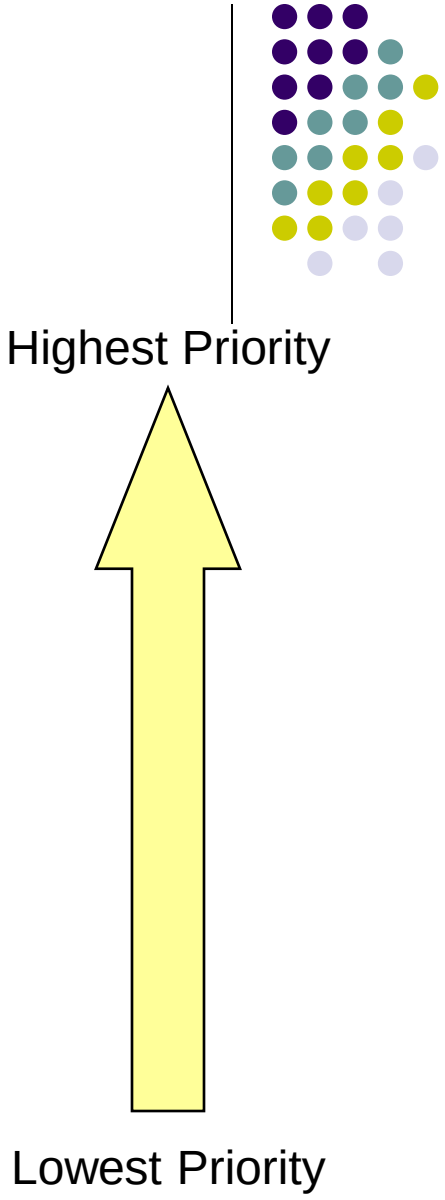
Smoothed Modules

- Smoothing eliminates discontinuities in the audio when changes are made by the control thread
- For ease-of-use, smoothing should be designed into the audio processing functions themselves and not be part of the control thread.
- Smoothing may be performed on a sample-by-sample (ideally) or block-by-block basis
- First order IIR smoothers are useful for many gain related changes and can be implemented efficiently (2 ops/sample on the SHARC)

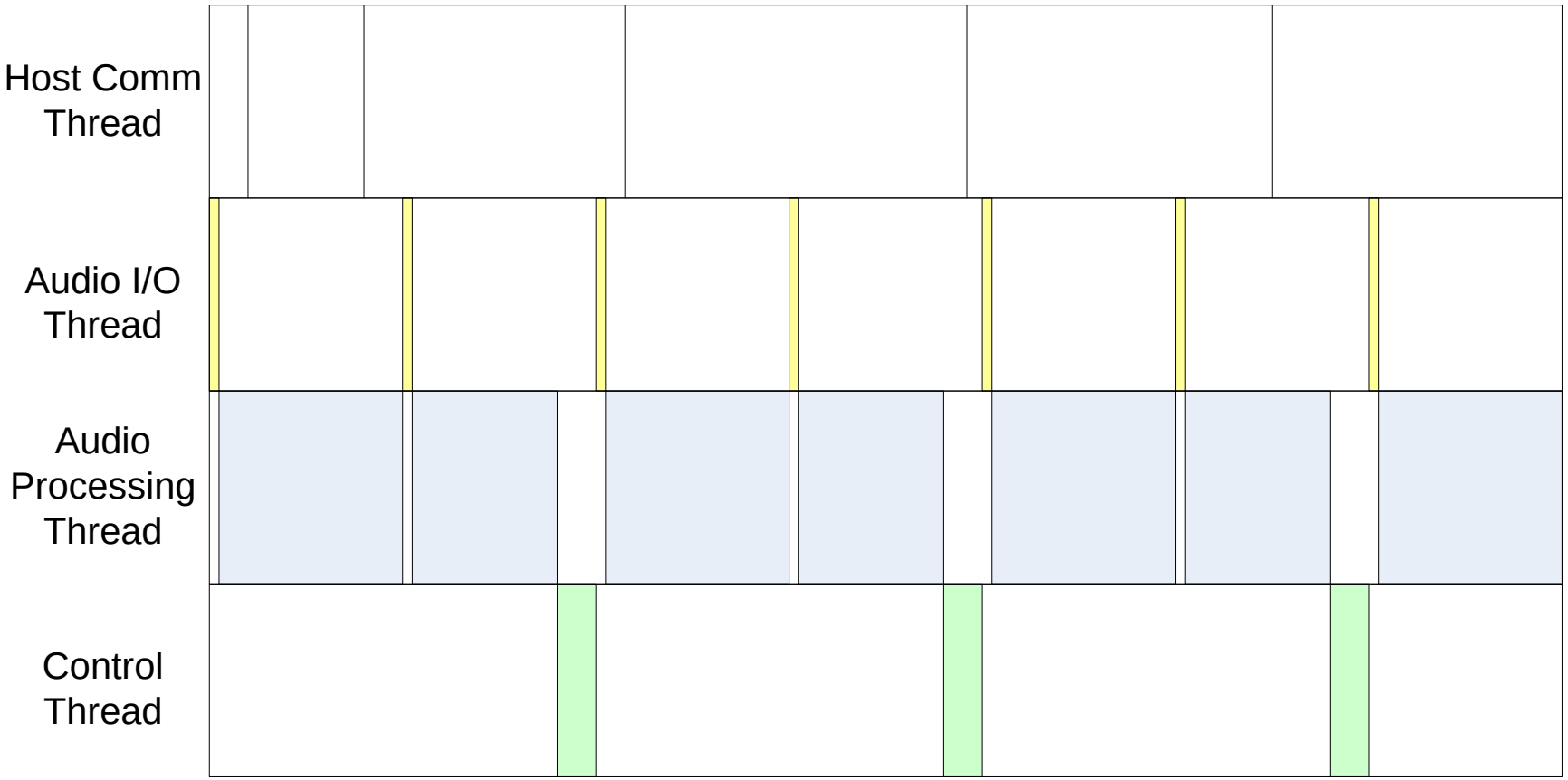


Basic Threading

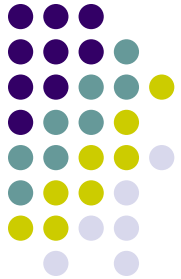
- Host communication interrupt
  - Asynchronous
  - Simple interrupt handler sends and receives data
  - Posts messages to the control thread
- Audio I/O interrupt
  - Uses chained DMA to reduce interrupt frequency
  - High priority prevents DMA from starving
  - Copies data in and out
  - Buffers up audio into larger blocks for processing
  - May perform audio process (if processing block size is the same as DMA block size)
- Audio processing interrupt
  - Performs all audio processing at a larger block size
- Control thread
  - All residual processing here
  - All control and housekeeping



Basic Threading Continued

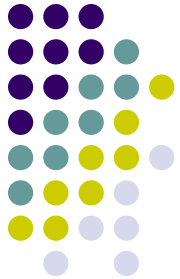


Is an RTOS Needed?



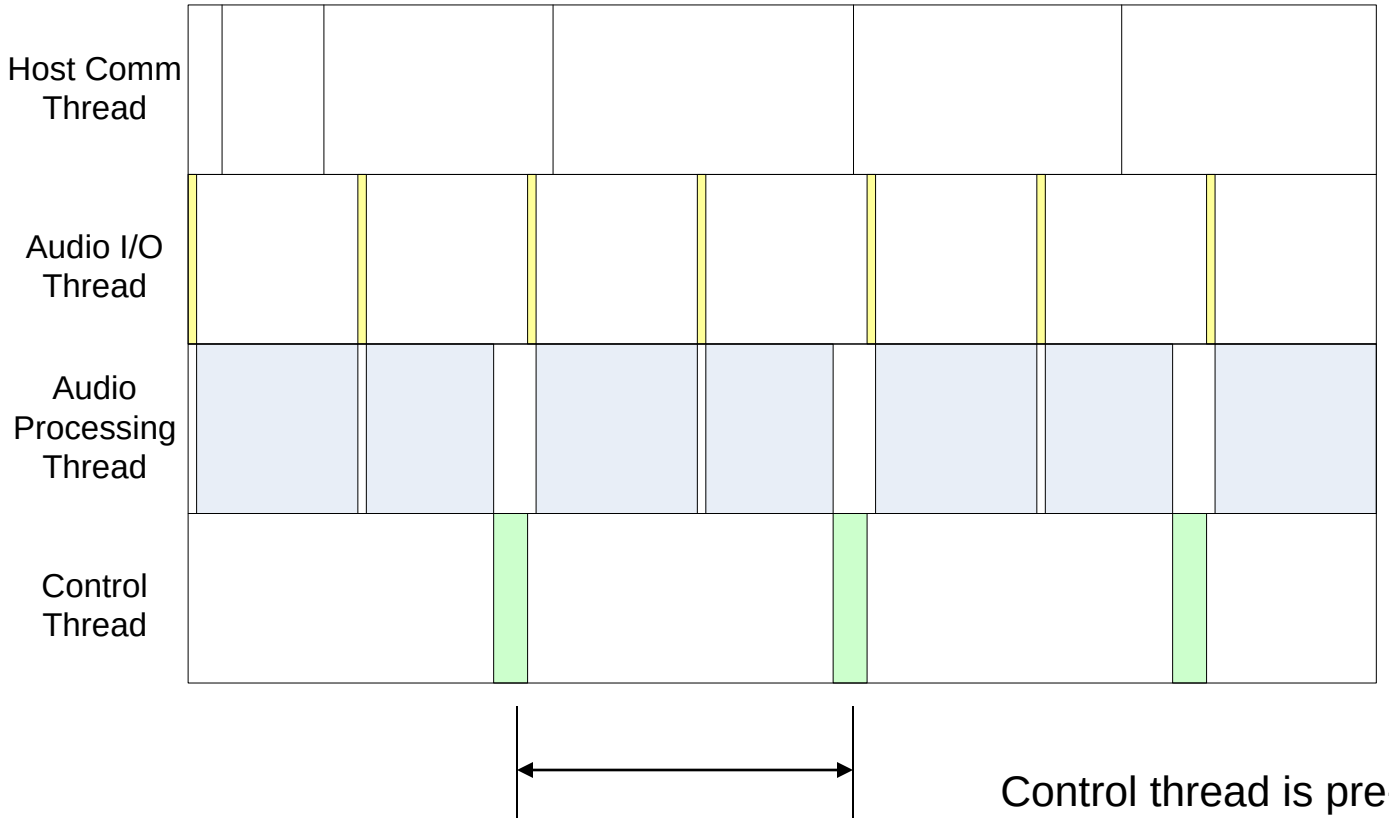
- Basic threading can be performed using interrupts and is sufficient for most product categories
  - Standard PCM processing (musical effects, live sound, professional audio, etc.)
  - Televisions
  - Musical instruments
  - AVR / decoders
  - Automotive amplifiers
- An RTOS is needed when there are other non-audio tasks being performed by the audio processor:
  - User interface
  - Ethernet stack
  - File system

Control Thread Issues



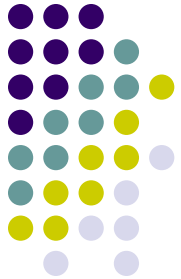
- The control thread must run frequently enough so as not to be sluggish
  - Psychoacoustic rate ~ 10 Hz
- Latency determined by the frequency of the control thread.
- The control thread will be interrupted by the audio processing - use critical sections (disable and enable interrupts) around sensitive issues.
  - Updating biquad filter coefficients
- Even a few percent of a modern DSP processor is sufficient for control

Control Thread Update Rate



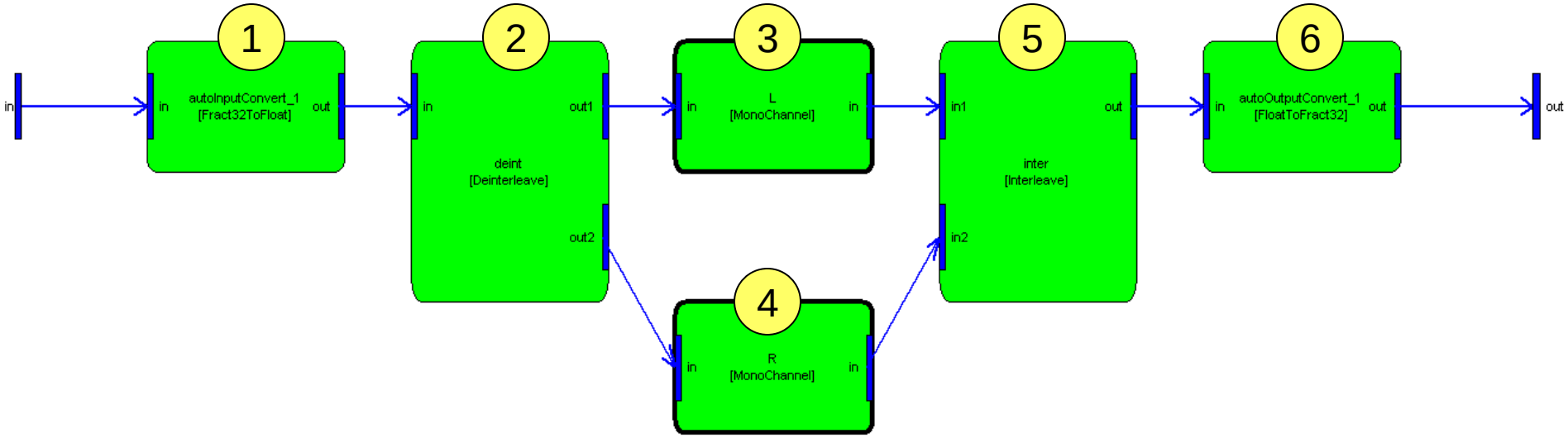
Control thread is pre-empted by the audio processing and essentially executes at the block rate.

Flexible Pin Types



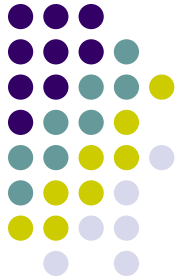
- Each pin is defined by the following pieces of information
  - Number of channels
  - Block size
  - Sample rate
  - Data type
  - Real or complex
- You can specify either *exact settings* for these values or an *allowable range*.
- Modules can be designed to operate on an arbitrary number of channels and block sizes
- One module can support many different data types. For example, the scaler module can operate on
  - Mono signals
  - Stereo signals
  - 5.1 signals
  - Control signals (1 sample)
  - Frequency domain signals
- Using ranges is critical to subsystem design and provides a great amount of flexibility and error checking

Pin Propagation



Subsystem drawings reflect the order of execution.

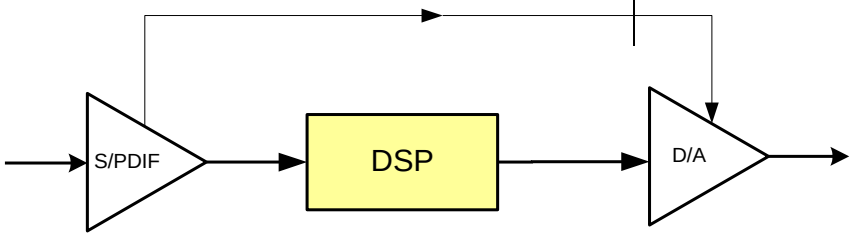
Sample Rate Dependent Processing

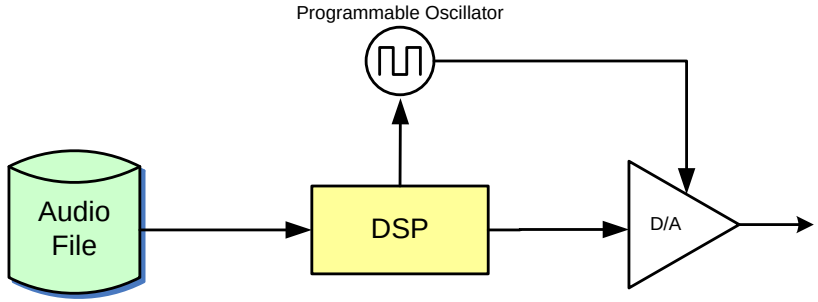


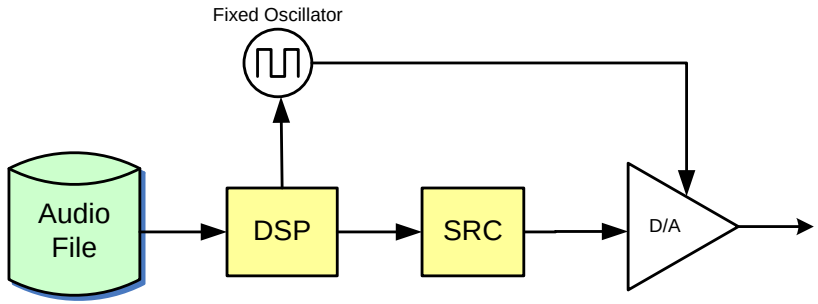
- The audio processing is sample rate dependent
  - Filter coefficients
  - Delay offsets
  - Sine wave frequencies
  - Time constants
  - Etc.
- The DSP needs to determine the sample rate
  - Meta-data in the file or data stream
  - Real-time measurements
- Handle sample rate changes by
  - Having design equations on the DSP
  - Storing sample rate dependent coefficients in tables
- Control thread must watch for sample rate changes and make updates accordingly

Decoder Integration

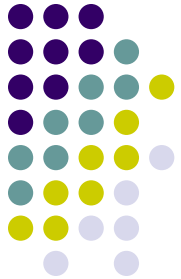
- "Push" style decoder
  - Data streams at a fixed rate over S/PDIF
  - Bitstream detector identifies stream type based on IEC standard
  - Output clock locked to incoming data rate
  - Dolby Digital and DTS
- "Pull" style decoder
  - Data read from a file or over the network
  - FIFO module buffers up input data
  - Variable bit rate decoding. Decoder pulls data from the FIFO as needed
  - File specifies the output sample rate. Clock must be generated or data resampled.
  - MP3, AAC, and WMA







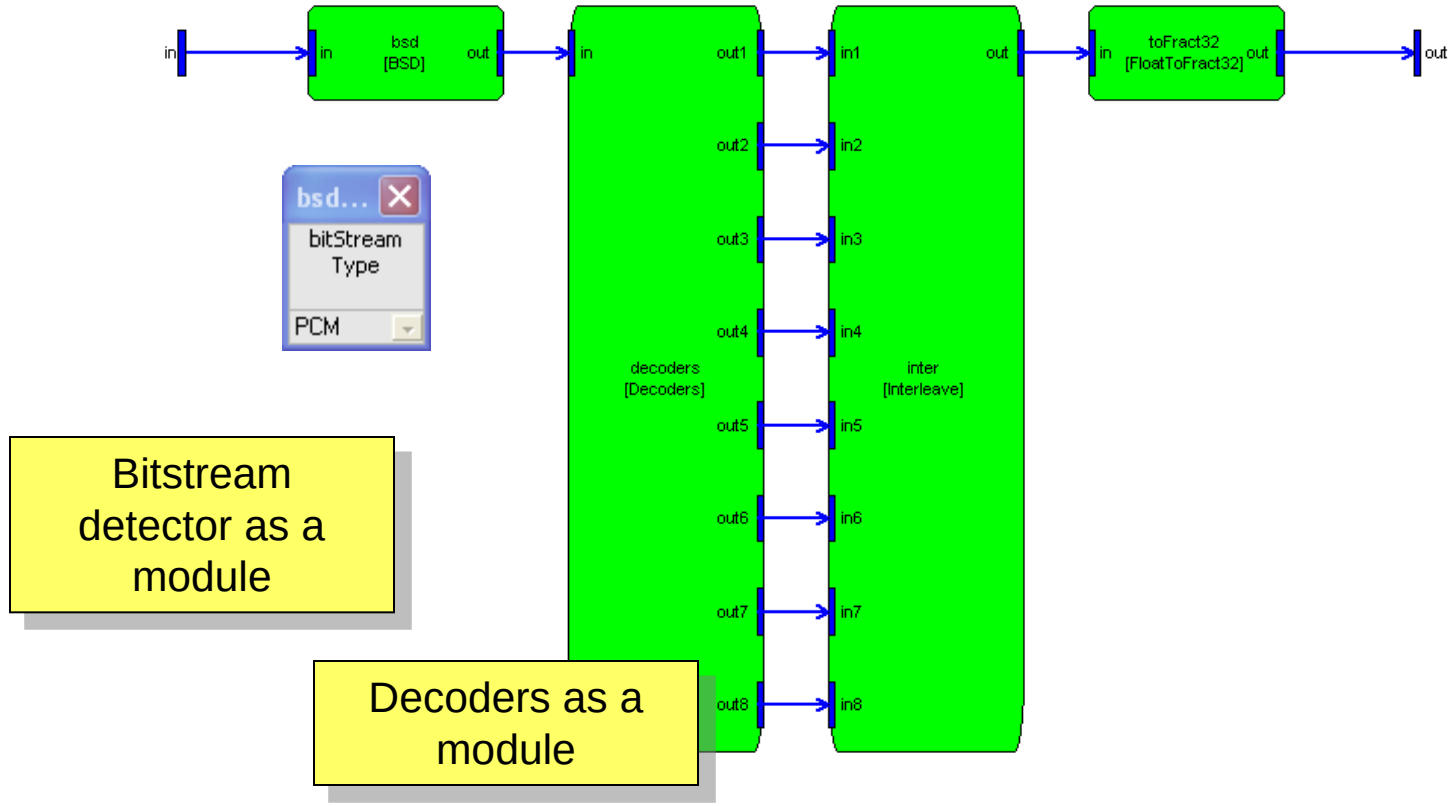
Block Sizes of Common Algorithms



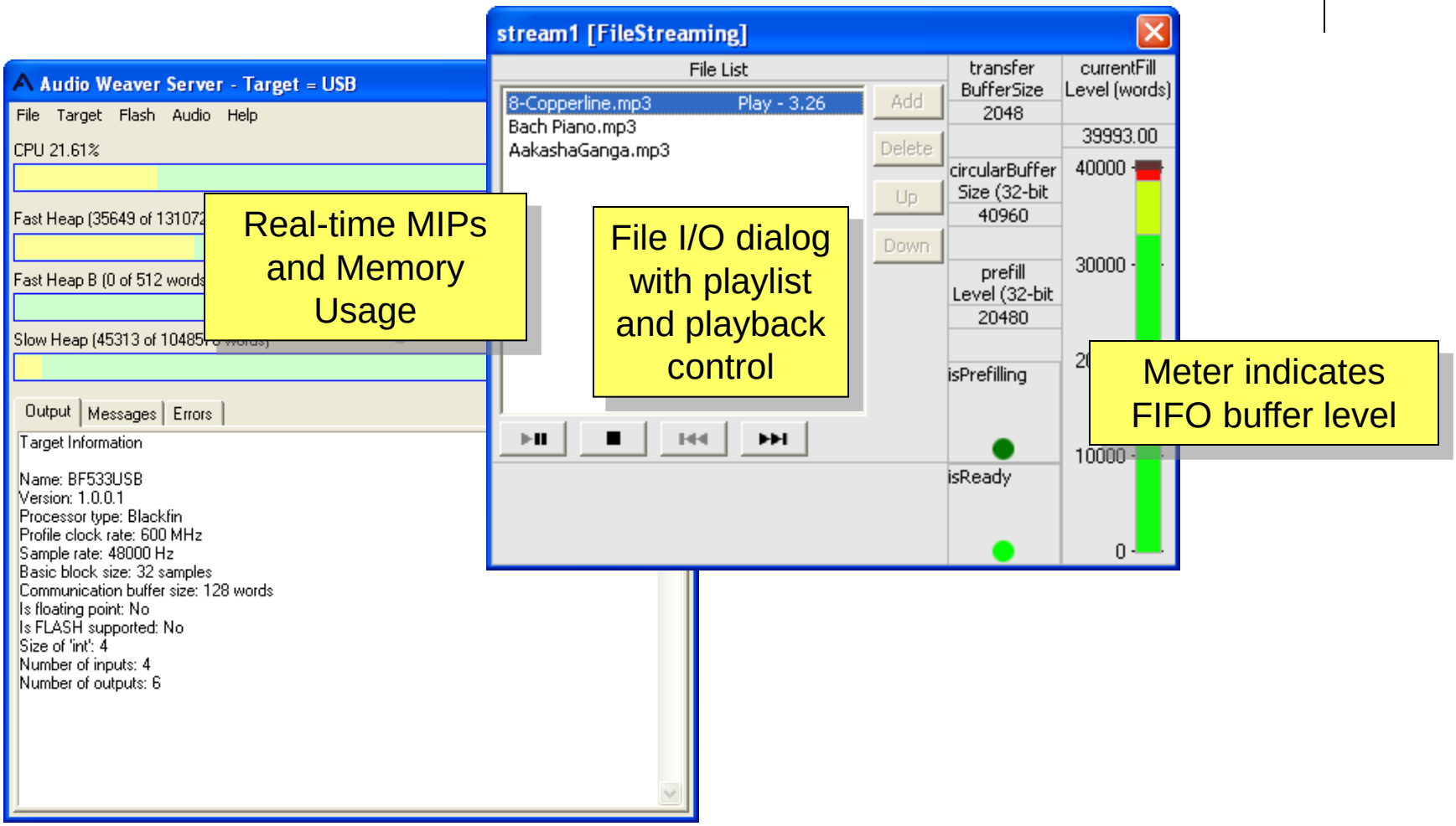
- Dolby Digital – 256 samples
- DTS – 256 samples
- MP3 decoder or encoder – 1152 samples
- AAC – 1024 samples

- Audio post-processing runs at a fixed block size (32, 64, or 128 samples)
- Decoder runs in a separate lower priority thread.

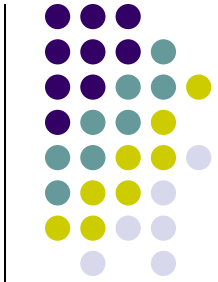
Dolby Digital Decoder



Blackfin MP3 Decoder



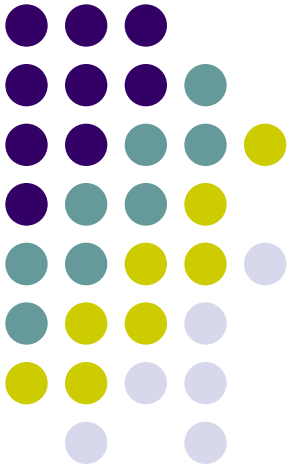
Summary



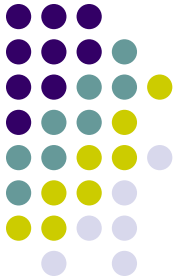
- Complexity of audio products continues to increase
- Software tools are the solution to manage the complexity
  - Provide a starting point and methodology for development
  - Automate common development tasks
  - Allow you to focus on product differentiation
- Tools will only become more important as processor capabilities continue to increase

Thank You!

Try Audio Weaver in Demo Room 133

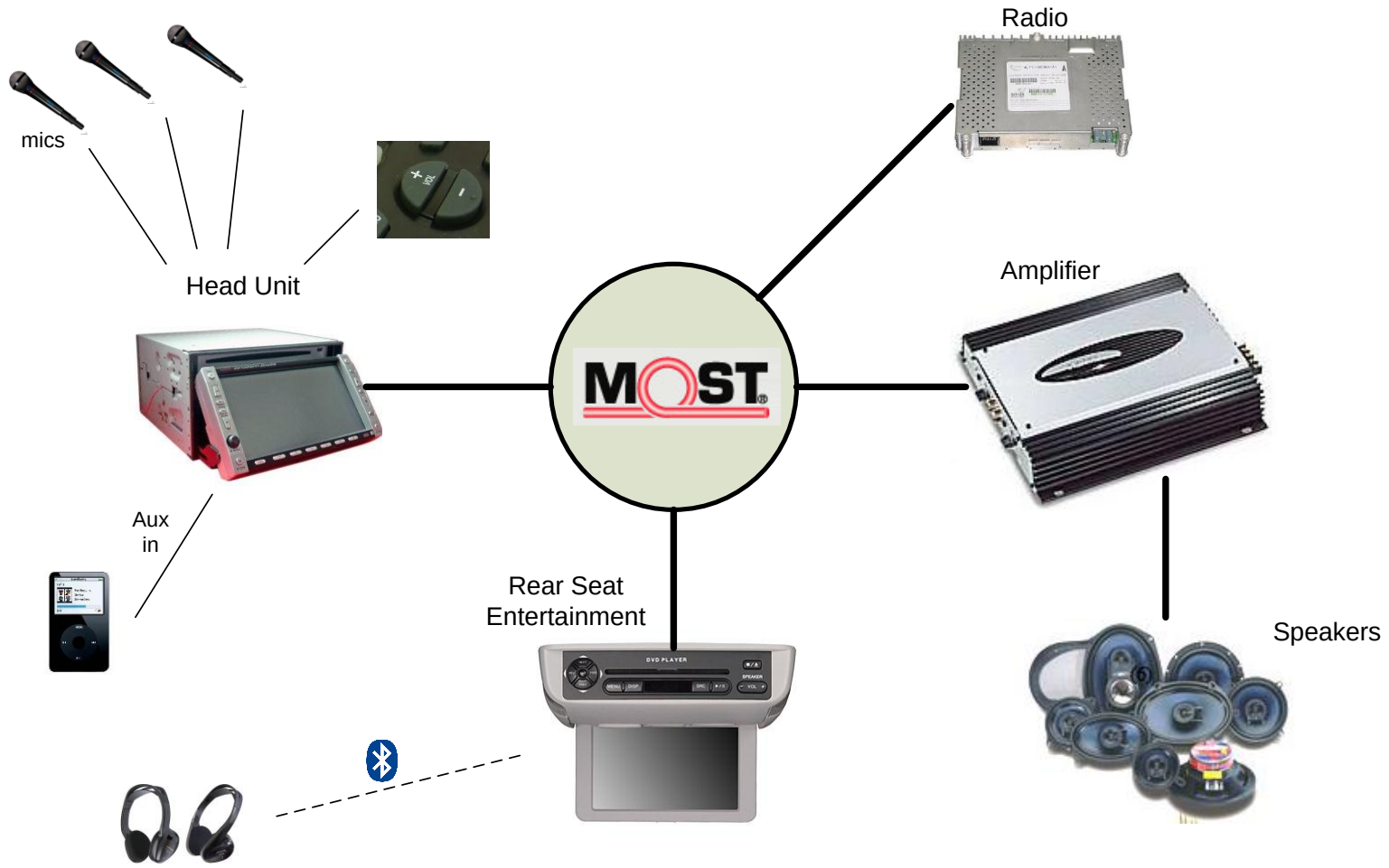


Audio Weaver Pricing

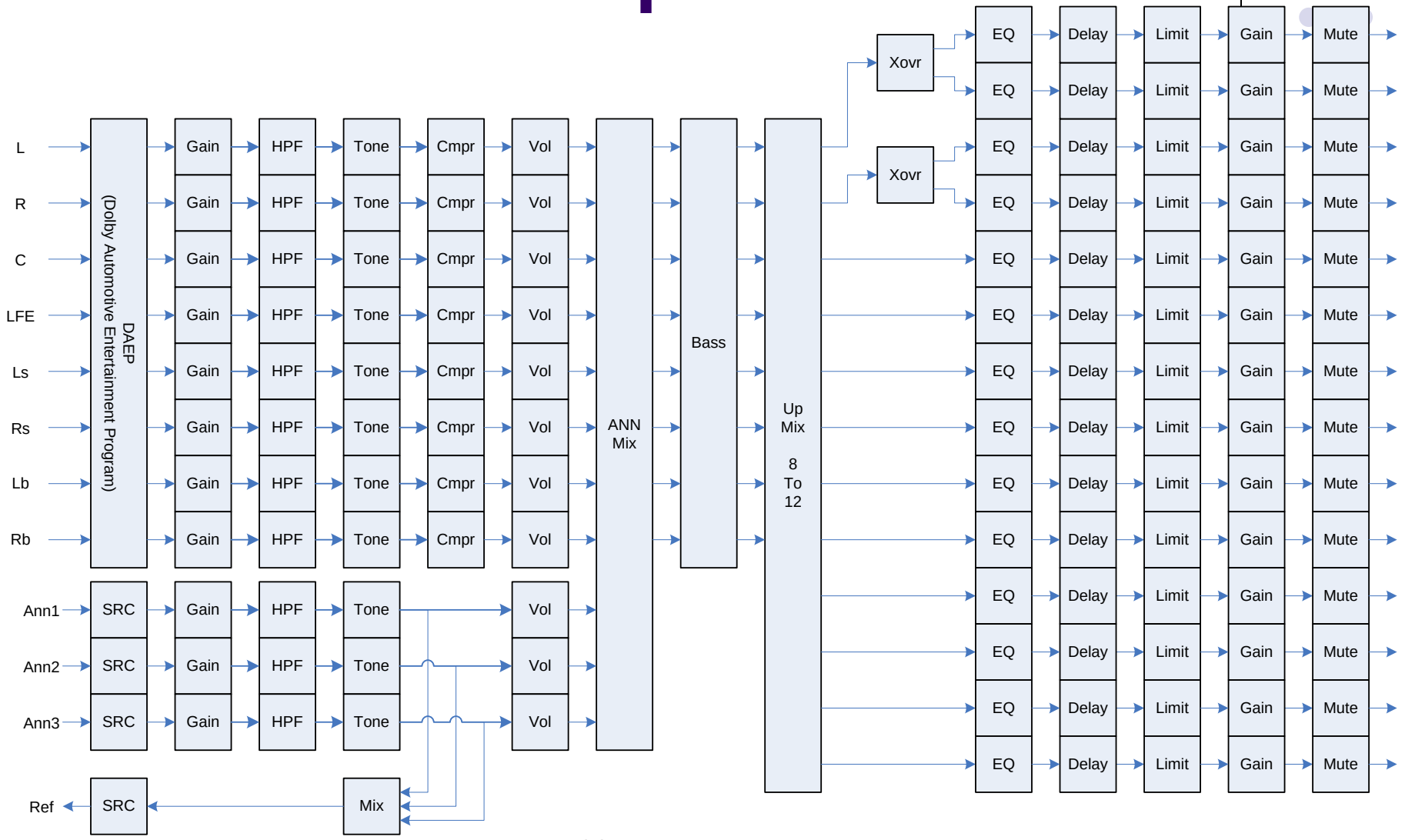


- Audio Weaver Designer - Free
  - Can be downloaded from www.dspconcepts.com
  - Create and tune audio systems on the PC or supported SHARC or Blackfin target hardware
  - Create custom audio modules on the PC. Create user interfaces
  - Requires MATLAB and VisualStudio (for module development)
- Audio Weaver Developer – USD $8,000
  - Create custom audio modules for an embedded processor
  - Create custom hardware targets
  - Requires VisualDSP++ and an emulator
  - Includes 20 hours of customer support or training
- Audio Weaver Production License
  - Ship products based on Audio Weaver
  - Licenses are based upon unit volumes.
  - Contact DSP Concepts for more information.

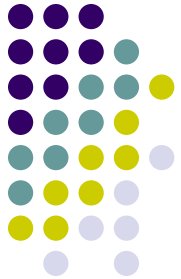
MOST System Architecture



Premium 14 Channel Automotive Amplifier

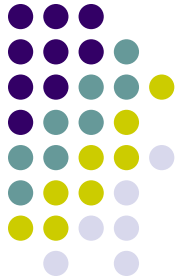


Announcement Channels



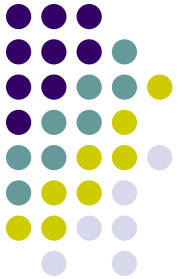
- Several monaural signals from non-entertainment sources
  - Navigation system
  - Telephone
  - Beeps/tones (turn signal, lights on, etc.)
- In MOST-based systems, signal arrive at a decimated rate (11.025 or 22.05 kHz)
- Sample rate convert up to 44.1 kHz and process along with the entertainment signals
- A “reference” channel is formed as the sum of the announcement channels and fed back to the echo canceller
- Some systems have a low-latency requirement on the entertainment channel and this can force operation at smaller block sizes.
  - Difficult to mix large block algorithms (DAEP) with low-latency processing

DAEP
Dolby Automotive Entertainment Program



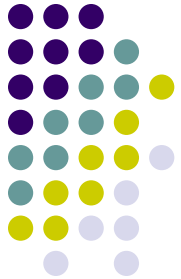
- Combines 4 different technologies
  - Prologic II – Converts all input sources to 8 output channels
  - eMix – intelligent signal combiner
  - Advanced Fader/Mixer – Handles front/back fading in the cabin
  - Phantom Center – Generates a phantom center image when no center speaker is present
- Solves the major automotive audio challenges
  - Spatial presentation without strong near side localization
  - Front/back fading. “Watch a movie in the rear seat”
  - Produces a strong center image – even when no center speaker is present.
  - Center channel can be panned to be in front of both the driver and the passenger.

Noise Compensation



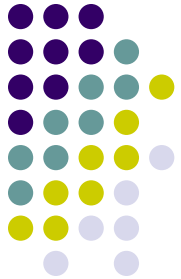
- Automatically adjust the volume control setting based on the noise in the cabin
- Noise estimate is based on
  - In cabin microphone(s)
  - Engine RPM
  - Vehicle speed
  - Climate control fan setting
  - Windows opened/closed
- Fairly common in premium systems
  - Bose “Audio Pilot”
  - Blaupunkt “Dynamic Noise Compensation”
  - Harman “Ambient Noise Compensation”

Hands-Free Technology



- Speaker phone integrated into the audio system
- Bluetooth connection to the cell phone
- Full duplex operation
- Adaptive filter for echo cancellation (speaker → microphone path)
- Other adaptive filters eliminate road, wind, and rain noise
- Some multi-microphone implement beam forming for improved intelligibility

Active Noise Compensation



- Microphones in the cabin measure the noise signal
- Control loop calculates “anti-noise” which is played back through the loudspeakers
- Extremely difficult to do in practice
- Requires low latency to be effective