



The TASP VSIPL Implementation

Some History
Goals and Limitations
How to Make It
How to Use It
How to Modify It

Randall Judd

SSC-SD

619 553 3086

judd@spawar.navy.mil



The Reference Version

- **DARPA**
 - VSIPPL was originated as a DARPA effort.
 - Hughes Research Lab (now HRL, LLC) was funded by DARPA to
 - Chair the forum
 - Write a reference version of the library
 - The HRL reference library
 - Was not designed for performance
 - Was only partially completed due to lack of funding



The TASP VSIPL implementation

- **NAVSEA**
 - NAVSEA was researching a common operating environment for signal processors.
 - This effort was called the Tactical Advanced Signal Processing Common Operating Environment.
 - The TASP COE team joined the VSIPL forum at its inception.
 - The TASP COE team needed a fairly complete reference implementation suitable for writing demonstration software.
 - The HRL development effort was picked up by TASP and the code rewritten to meet the needs of the TASP group.
 - Including modifications to keep up with changes to the specification



TASP VSIPPL Development Goals

- **Dependent only on an ANSI C compiler**
 - Be a portable implementation that runs on all platforms as a baseline.
- **Enough performance to be usable for demonstrations and code development work**
- **Be compliant with the VSIPPL Specification**
- **Be usable as a starting point for Vendor implementations**
- **Be complete**
 - Missing 2DFFT, 3DFFT, SVD, and a few other things.
- **Support both defined library modes**
 - **Development**
 - **Not currently supported**
 - **Production**

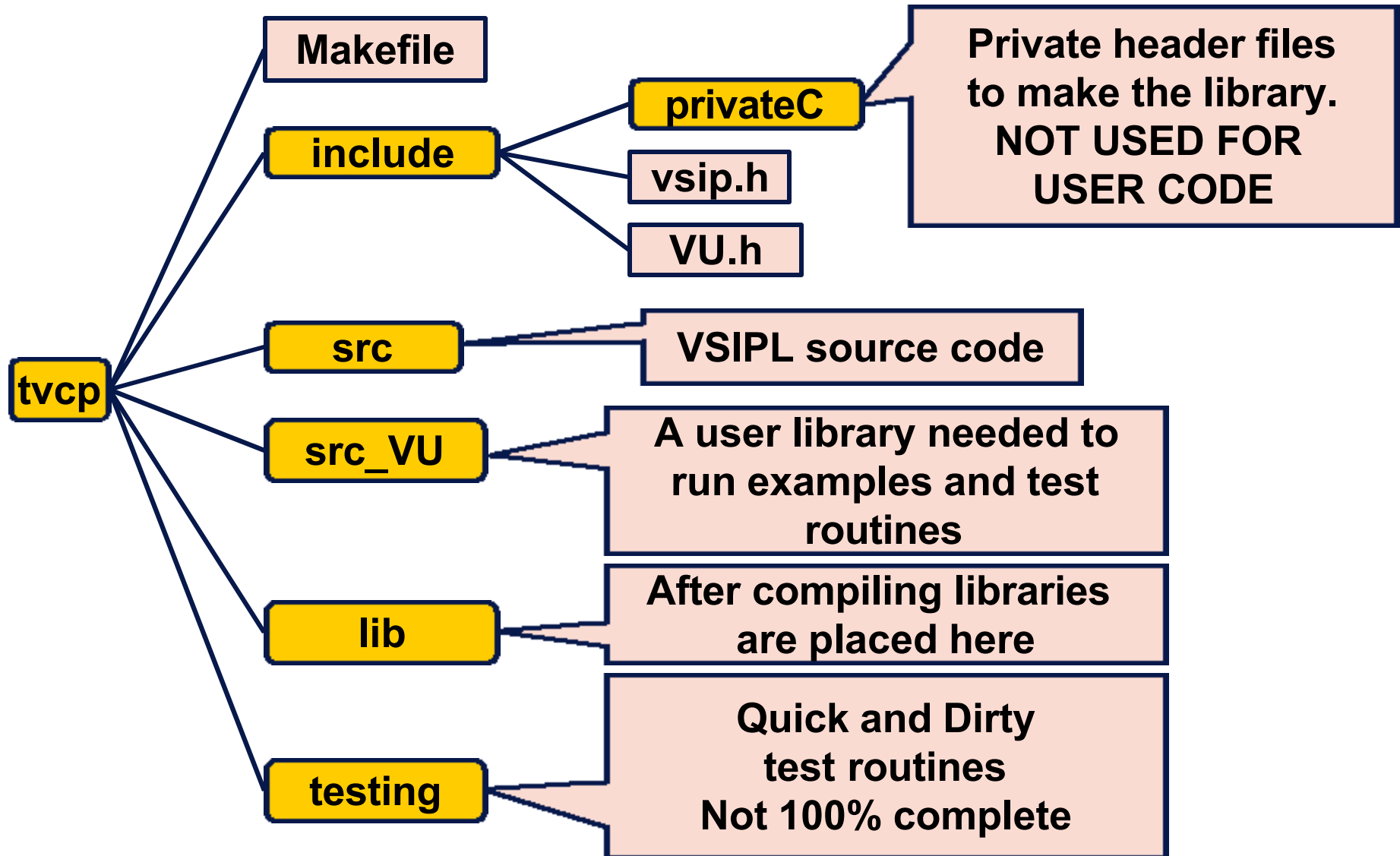


TASP VSIPPL limitations

- **The implementation is NOT part of the specification**
 - The VSIPPL specification is a standalone document.
 - Differences between the specification and the TASP implementation may exist.
 - If you find some let me know
 - When defining VSIPPL functionality the specification is the only authority.
- **No attempt is made to get the optimum performance from any platform.**
 - Portability and functionality are the main goals
- **Thread safety is not built in**
 - I would need to use a thread library which would make the code less portable.

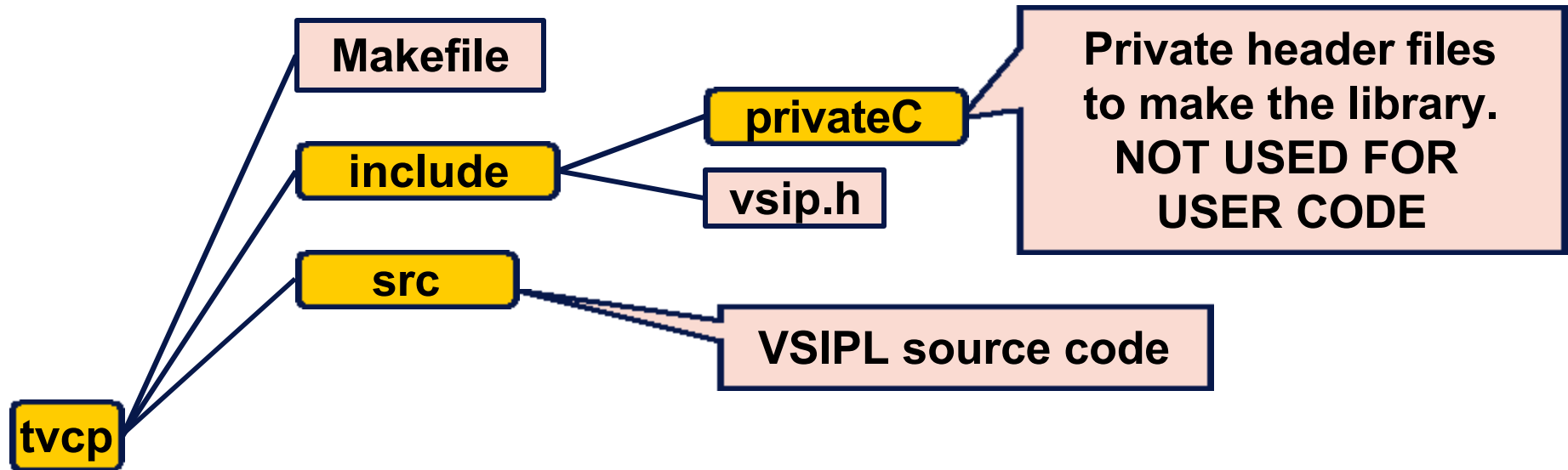


Implementation Layout for Core Plus





Making the Library



- **For a standard GNU environment the Makefile will probably work as is. Just type “make”**
 - There is no configure option. Modify the Makefile manually for different compiler options.
- **If you don't want to or can't use the make utility then just include all the source files, and header files in whatever compiler tool you prefer.**



Using the Library

- **Copy the VSIPPL library archive to whatever library directory you prefer.**
- **Copy the VSIPPL header file to an appropriate include directory.**
- **The TASP VSIPPL Implementation depends upon the standard C math library.**
- **I have tried to minimize internal dependencies (where one VSIPPL function depends on another), but there are a few.**
 - **You only need to worry about this if you want to build a custom profile.**
 - **To build a custom profile take the functions you know your need and let the compiler tell you what is missing.**



Modifying the Library

- **The key to modifying the library is to understand the support functions.**
 - How blocks work.
 - How views reference blocks
- **Everything else is just pointer math.**
 - Basically use the view to get the block.
 - Use the block to get the pointer to data
 - Use the view to find strides, lengths and offset needed to access the data.
- **The only complication is complex blocks**
 - Complex blocks have an internal stride to handle split or interleaved data.
 - This stride is in the block, not the view, and is hidden by the implementation.



Summary

- **TASP VSIPPL is an open source ANSI C implementation of the VSIPPL production mode specification.**
- **Enough functionality has been developed to support the entire core profile.**
 - Additional functionality is available in the core plus distribution.
 - Missing functions are primarily Multidimensional signal processing functions and the SVD from the linear algebra section.
- **The library has been used enough that it is relatively bug free.**
 - There are always some bugs. We just don't know where they are.

QUESTIONS?