

# About These Slides

These slides were written for the term project presentation in Fred Martin's 91.548 Robotics 1 Spring 2006 course.

They explain the very basics of what Phission is about for those who weren't previously familiar with Phission. The slides proceed to discuss the process of porting Phission to the Blackfin/VDK OS in the VisualDSP++ IDE. Also discussed is work on the OV7620 camera capture class that integrates with Phission's architecture, how Phission is tested in VDSP++, an example of some problems encountered, the final status of Phission on the Blackfin/VDK OS and a couple future enhancements.

# 91.548 Robotics Term Project

## Phission on the ADI Blackfin & UML HandyBoard

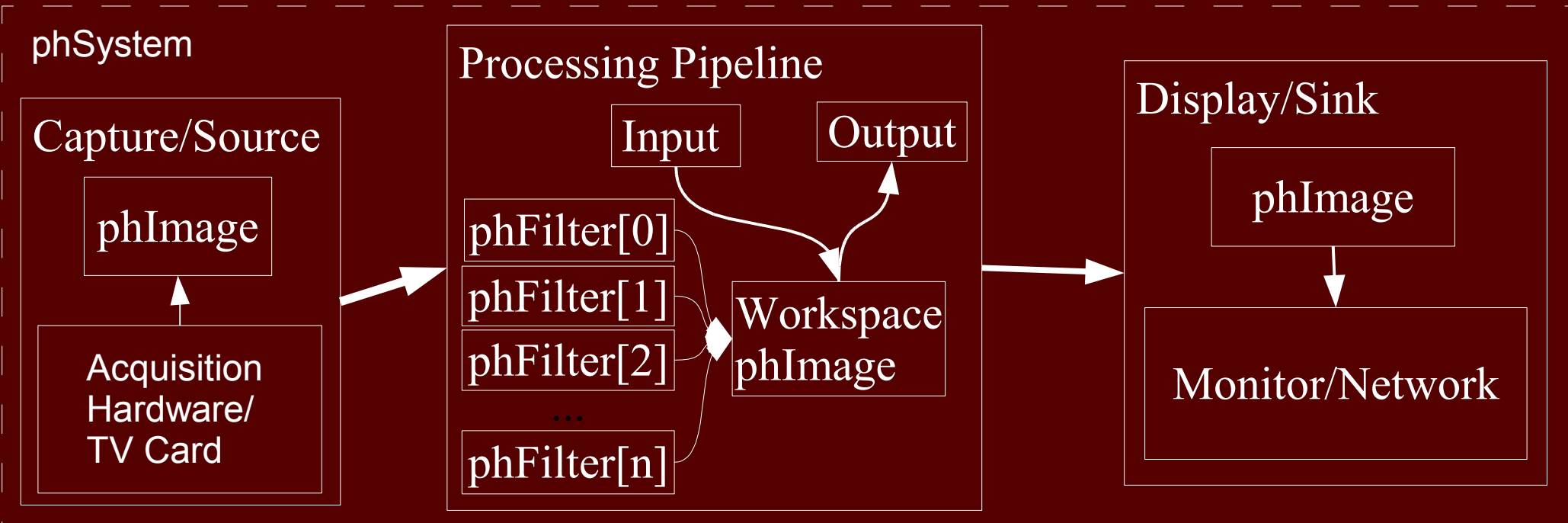
# Presentation Outline

- What is Phission?
- Why port to VDK Blackfin Handy Board?
- Integrating Phission into the VisualDSP++ IDE
- Port Phission OS classes to the VDK API
- The OV7620 capture class & problems
- Testing Phission & some problems
- Port status
- Future Enhancements

# What is Phission ?

- Phission is a portable concurrent modular vision processing system toolkit
- Used to construct a software vision processing sub-system in parallel with the main mobile robot control loop
- Simple capture -> filter/process -> display data-flow style architecture

# What is Phission ?



# What is Phission ?

- Supplies portable OS level primitives
  - phMutex
  - phSemaphore
  - phCondition
  - phRWLock
  - phSocket
  - phThread
  - phTimeStamp

All operating system dependent code is encapsulated within these classes

# What is Phission ?

- Supports common capture interfaces  
Video4Linux,  
VideoForWindows,  
AvCodec supported video files &  
Network sources
- Supports common display interfaces  
SDL, X11, GDI, FLTK and Network sinks

# What is Phission ?

- Supplies the common image filters gaussian/median/mean blur, canny & sobel edge detection, histogram, segment/blob, etc.
- Only operates on 8 bit image formats

# VDK Blackfin Handy Board System

- VisualDSP++ IDE as development platform
- VisualDSP++ Kernel (VDK) API as the operating system
- OmniVision 7620 video camera source
- PPI / TWI interfaces for acquiring data and controlling the camera
- On chip Ethernet interface accessed through Lightweight IP(LwIP) library

# Why Port Phission to VDK/Blackfin ?

- 600MHz low power embedded system is powerful enough for vision processing
- Provides Memory-to-Memory DMA
- Has Parallel Peripheral Interface DMA that captures high res. images at high frame rates w/o CPU overhead
- Blackfin is the new Handy Board processor
- Minimal hardware resources usually inspire innovation in software
- Phission code will work the same on your desktop and the Blackfin w/few exceptions

# Integrate with VisualDSP++ IDE

- Created a Phission Library Project
- Includes all files except unsupported APIs; i.e. SDL, GDI, V4L, Vfw, etc.
- phStdint.h to define all the stdint.h types
- Static phission.h & phissionconfig.h files for the VDK (these are usually output from the configure script)

# Integrate with VisualDSP++ IDE

- `#ifdef HAVE_HEADER_H` for all system header files; some may not be present
- `#ifdef`'d out all file writing support
- Added `LITTLE_ENDIAN` conditional definition to `phByteOrder.h`

# Integrate with VisualDSP++ IDE

- Create a distributable Phission library output from the VDSP++ IDE
- Use the VDSP++ Automation engine to create GenerateHeaders.js which creates the library's distribution include/ directory
- Use Post-build command to copy current Debug/Release library file to distribution lib/ directory
- Application code only needs to provide the path to the VDK include/ & lib/ directories for compiling and linking

# Integrate with VisualDSP++ IDE

- Create a libjpeg project and port IJG's jpeg library to the VisualDSP++ IDE
- Similar include/ & lib/ directory outputs for distributing the library
- No alterations to the code base required because libjpeg is all C source code
- Created the jconfig.h header file to define the system attributes for the jpeg code
  - jconfig.h is mostly the default header

# Integrate with VisualDSP++ IDE

- Create a libjpeg project and port IJG's jpeg library to the VisualDSP++ IDE
- Similar include/ & lib/ directory outputs for distributing the library
- No alterations to the code base required because libjpeg is all C source code
- Created the jconfig.h header file to define the system attributes for the jpeg code
  - jconfig.h is mostly the default header

# Port OS Classes to VDK API

## *phMutex*

- The first attempt at porting used the VDK::SemaphoreID related functions
- The second attempt used VDK::Push/PopUnscheduledRegion loops with an “is locked” variable
- The final implementation required communication with ADI's Project Tools Support group to get an unreleased VDK::RMutex API

# Port to VDK API

## **phSemaphore**

- just wrapped the VDK interface

## **phCondition**

- not natively supported by VDK OS

Uses a semaphore/mutex/FIFO combination to create the condition construct

## **phRWLock**

- not natively supported by VDK OS
- Uses condition variables and mutexes to create the reader-writer construct with reader/writer regions of access

# Port to VDK API

## *phThread*

- VDK threads can be dynamically created
- The templates to create threads require a VDK::Thread derived class
- phWrapperThread is a friend class of phThread to permit it to call phThread::entry
- phWrapperThread is defined in the source file to preserve the encapsulation of OS APIs

# Port to VDK API

- Time routines for the VDK work in “ticks”
  - convert seconds/microseconds/etc. values to ticks using `VDK::GetTickPeriod()`
  - Other code uses `CLOCKS_PER_SECOND` to calculate seconds/etc. values
    - This needs to change because `CLOCKS_PER_SECOND` is a compile time setting and tick period values could change during runtime
- VDK supplies the `clock.h` interface as well

# Write OV7620 Driver

- Example camera code and project was given by ADI
- Ported all the register information and operations of the OV7620 to replace the OV74xx driver
- Developed all initial acquisition code in a standalone program that uses the PPI port and TWI interface
- Verified camera capturing using the VDSP++ Image Viewer

# Write phOmniVisionSource class

- Derives from the Phission phImageCapture C++ class interface like the other capture/source objects(V4L,VfW, AvCodec,etc.)
- Ported code from standalone into the phOmniVisionSource class
- Outputs an RGB24 image from the Bayer Matrix formatted RGBG camera output

# phOmniVisionSource: Problems

- System initialization code overwrote Ethernet interrupt settings and prevented Ethernet/LwIP from working
- Images obtained from camera module aren't correct
- The Bayer Matrix format output is RGBG which represents 2 pixels and requires CPU overhead to reformat for RGB 24bit output

# Testing it

- Wrote an application to test pieces of Phission that were being ported
  - phThreadTest, phRWLockTest, BlobTest and phOmniVisionTest
- The test code uses two boot threads
  - ***lwip\_sysboot\_threadtype***
    - initializes ethernet stack & threads
    - obtains IP address with DHCP
  - ***PhissionBootThread***
    - Waits for ethernet initialization
    - Sets up argc/argv parameters
    - Calls the test's main function

# BlobTest Problems

- BlobTest verifies that the segmentation/blob code works
- Used GIMP to create static C files with image buffers
- Images contained red blobs and were segmented
- Stack issues caused a rewrite of the Quicksort code (which sorted by blob size)
- The blob data is sorted with an iterative Quicksort

# Blackfin Port Status

- phOmniVisionSource class needs to be debugged
  - OV7620 driver code and capture class haven't been released yet due to pending questions on license and copyright issues
- Phission runs on the Blackfin processor with the VDK operating system
- There is code that demonstrates using Phission on the Blackfin by capturing from the OV7620 and sending JPEG images using the NetDisplay/phNetSource combination

# Future Enhancements

- Perform memcpy using the Memory-to-Memory DMA
- Reorganize Bayer Matrix/RGBG using the FPGA or some Blackfin vector optimized routine
- Get a Blackfin optimized JPEG compression library