Chandra Library User's Guide

Martin Jay McKee

July 12, 2020

Contents

1 Overview

Chandra is a collection of almost entirely header-only libraries that are designed to support the development of real-time embedded control systems on microcontroller systems. While Chandra sup-

ports running inside an RTOS, it does not require it because the libraries are designed in such a way that functionality can easilby be called from an explicit event loop.

1.1 Description

As they stand, the Chandra libraries consist of four sub-libraries: Chandra-Core, Chandra-HAL, Chandra-Control, and Chandra-Aero. The Chandra Core libraries are functionality that is required for any of the other libraries but which is - otherwise - stand alone. The core libraries could be used completely by themselves, though they do not provide enough functionality to implement a complete application because they do not contain features such as input and output. The HAL libraries provide a hardware abstraction layer for the platform with a consistent API. This is the only sub-library that requires explicit porting to new platforms[1]. The HAL includes things such as a system clock implementation, timing functionality, GPIO, serial peripherals, etc. Some of the functionality in the HAL is platform independent as it is based directly on lower level functionality. The porting documentation later in this guide describes what types and functions need to be implemented. The third sub-library is Chandra-Control which provides observers, filters and control loop implementations. Finally, Chandra-Aero provides special-purpose functions for aerodynamics related programs. This includes functionality such as an atmosphere (altitude, pressure, temperature, density) model, basic air properties, lift and drag calculation, etc.

1.2 Dependencies

The Chandra libraries are written against the C++14 standard and require a compiler which is compatible with the standard,

[1] In future, it may make sense to optimized some of the core functionlaity based on the platform. This might include chosing loop variable sizes or other microoptimizations. The core functionality should work without requireing any porting, however.

including language features such as: ranged for loops, generalized lambda expressions, constexpr expressions, and variadic templates. Any C++14 compliant compiler should have no issues with the code. Only minimal standard library support is needed but includes: chrono, type_traits, number_traits, and std::common_type.

To use the code generators bundled with Chandra, it is also necessary to have a Python 3 interpreter installed along with several libraries. To run the generator for Chandra Units, the Mako Templating engine is required. For the Kalman Filter Optimizer, Mako and SymPy are required.

The documentation is built in LaTex and requires a number of packages including pygments (which - in return - requires Python) and graphicx. In general, users should not need to rebuild the documentation as Chandra is distributed with the final built PDF of the documentation.

1.3 Platform Indepedence

The bulk of the Chandra libraries are platform independent. The only portion of the libraries which are platform dependent are those within Chandra-HAL – the *H*ardware *A*bstraction *L*ayer. Porting of the HAL functionality to new platforms is required but the remainder of the libraries can be used with no changes so long as a C++14 compiler and standard library are available. If some standard library features are unavaliable on the target platform, they can also be included externally.

1.4 User's Guide Organization

This guide is arranged in five major sections. The first, this overview, is simply a highlevel overview which should be sufficient for a new user to understand what the Chandra library is - in a basic way - but ability to use the library should not be expected by this point. The following four sections are intended to provide a deeper introduction to the components in the library, as well as rationale, usage information, and general reference. The second

section of the guide is a getting started chapter which inlcudes ?? complete applications (targeted at an NXP LPCXpresso1549 development board) to demonstrate basic use of the library, features, code structure, and compilation. The third section of this guide covers the overall library architecture. It introduces all the major components of the library and the rationale for including them. Following the architecture documentation, the is an in depth API (*A*pplication *P*rogrammer *I*nterface) reference which describes the complete interface for every type and function. The final section of the guide is a code "cookbook" which includes code fragments to solve specific problems. This final section is intended to simplify the learning of the library as well as demonstrating some of the less obvious features.

Beyond the primary descriptive sections

Part I

Library Introduction

2 Getting Started

The Chandra libraries were originally written for personal projects targeting the NXP LPC series of ARM Cortex-M microcontrollers. Despite this, however, the Chandra libraries were writen from the very beginning to expose an interface that can be ported to multiple architectures without loss of functionality. The Chandra libraries are designed to e extremely powerful without being too excessively difficult to use. While they may not be as simple to use as an Arduino compatible platform, the libraries are designed in such a way as to provide more direct access to the hardware and so that they can be optimized[2] to a much higher level than some of the core Arduino functionality. Having said that, Chandra is not a competitor to Arduino. It is aimed at implementing control systems for robotics. Specifically, it is designed for things like rovers and flight computers. Additionally, while it could certainly be used for high-level systems, it is mostly intended for deeply embedded, real-time systems.

This section will introduce some of the central concepts and functions in the Chandra library in the context of example applications.

[2]Compilation of an application built on the Chandra Libraries without compiler optimization *WILL* lead to large slow programs. Chandra is implemented using templates and other advanced C++ features that cause bloated unoptimized code. Nevertheless, when compiler optimization is enabled, much of the functionality is optimized to very nearly the hand-written equivelent. Do not compile Chandra programs without optimization and expect svelt code. You won't get it. With optimization, however, you will.

2.1

3 Library Architecture

3.1 Overview

The following sections provide an outline of the library facilities. Each feature will be explained and a very basic example of use provided. Complete description of the API is saved for the API chapter which is arranged in a manner that mirrors this chapter. Essentially, this chapter is an introduction to functionality and the rationale behind it while the API reference gets into the nuts and bolts of each function, class, and datatype provided by the library.

3.2 Core

Math

Matrix Types

The ADD SYMMETRIC AND SKEW-SYMMETRIC MATRICIES. THESE CAN HAVE OPTIMIED STORAGE AND MAY BE POSSIBLE TO OPTIMIZE OPERATIONS FOR THESE SPECIFIC FORMS (I.E. ADDITION CAN BE OPTIMIZED, MULTIPLICATION THAT CREATES A SKEW-SYMMETRIC MATRIC MAY BE OPTIMIZABLE, ETC.

Matrix Operations

IT MAY MAKE SENSE TO USE EXPRESSION TEMPLATES TO SELECT THE IMPLEMENTATION OF OPERATIONS.

Units

3.3 HAL

Chandra::Chrono

Sysbem Clock

System Frequencies

Performance Timers

General Purpose Input/Output

Analog Input/Output

CURRENTLY THERE IS NO API SPECIFIED FOR DAC, THERE SHOULD BE

Communications

I^2C

SPI

USART

Drivers

Barometric Pressure Sensors

Inertia Measurement Sensors

Servo Drivers

3.4 Control

Kalman Filtering

The Kalman filtering components of the Chandra library are designed to support multiple approaches to constructing a kalman filter. Explicit implementation of a Kalman filter directly in source code by configuring the various matricies is simple. This also has the advantage that it allows for dynamic modification of the matricies during runtime.

Code Generator

PID

Builtin Estimators

Madgwick's AHRS

3.5 Aero

ATMOSPHERIC PROPERTIES, MODEL OF ATMOSPHERE (PRESSURE, DENSITY, TEMPERATURE, ALTITUDE, ETC.), BASIC AERODYNAMICS

Part II

Library Deep Dive

4 API

4.1

5 Chandra Cookbook

5.1 Clock and Timing

Performance Timers

Timing a function

Conditional timing of a function

Loop frequency measurement

Loop frequency statistics

5.2 Digital I/O

Reading a digital pin

Writing to a digital pin

Reading a mechanical switch/button

Manual control of an LED

Automatic PWM of an LED

Automatic PWM of an LED with asynchronous update

Event on a pin falling (rising) edge

Reading a Quadrature Encoder

5.3 Analog I/O

Part III

Appendicies

A Porting Guide