**Software Architecture Specification**

**Small Volume Ink System**

**Contents**

1. **Introduction**
2. **System Context**

2.1 System Boundaries

2.2 Users

2.3 External Systems

1. **Architecture Overview**

3.1 Producer/Consumer

3.2 State Machine

3.3 Program Routines

3.4 Flow Chart

1. **Functional Architecture**

4.1 Components

4.2 User Input & Interface

4.3 Monitoring System

4.4 Main Controller

4.4.1 Pressure

4.4.2 Temperature

4.5 Hardware Control Interface

4.5.1 Pressure

4.5.2 Temperature

4.6 Data Logging

4.7 Termination & Error Handling

4.8 Optional Blocks

1. **Architecture Design Decisions**

5.1 Modular Design

5.2 Timing Considerations

**1. Introduction**

The Small Volume Ink System is conceptualized to address the need for a more efficient, analyzable ink delivery system. The physical components will consist of two main vessels from which ink continually circulates, sensors to acquire information, and components to manipulate the system environment.

The project is tailored to be completed over a 3 month period, with considerations of further development/implementation.

**2. System Context**

**2.1 System Boundaries**

There are many different factors that exert an influence on the software system. The system boundaries are outlined in below to give a high level model.

Figure 1 – Context diagram of software system

C:\Users\v_wong\Downloads\System Context Diagram ff.png**2.2** **Users**

The software system will involve two main roles associated with the users, that of an operator and that of an analyst.

The operator interacts with the system as part of a larger system to complete a mechanical task.

“*As an operator, I want to be able to provide continuous ink delivery to the printhead with a small volume so that I do not spend resources preparing large volumes of fluid.”*

The analyst interacts with the system with the small volume ink system to ascertain information regarding the fluid being used.

*“As an analyst, I want to be able to obtain diagnostic metrics from the system so that I can further development with the information.”*

**2.3 External Systems**

The user interface will be centralized on a PC/Laptop running the LabVIEW program. User commands will be interpreted as a series of keystrokes or mouse clicks.

The software system will also interface with a data acquisition device (DAQ) as a means of communicating with hardware circuits for inputs and outputs.

Reference to hardware specification document? Yet to be written

**3. Architecture Overview**

**3.1 Producer/Consumer**

The overarching design for this architecture is that of a producer utilizing multiple, parallel consumers. This allows user commands to be put into a queue while previous operations are being processed. The disjointed nature between work orders and processes can be taken advantage of, allowing individual timing considerations and simplifying integration of extra functions.

**3.2 State Machine**

In addition to the aforementioned producer-consumer pattern, the software architecture also utilizes state machines within the producer and consumer loops. The various states are designed the handle different modes of operation, such as program routines, manual control or PID control.

**3.3 Program Routines**

These will be either static protocols running predetermined processes, such as initialization, shutdown or in the context of ink systems – a flush function, or dynamic looping processes such as PID control of pressure/temperature.

More on routines needed

**3.4 Flow Chart**

**Flowchart diagram + more**

**4. Functional Architecture**

Lots of software screenshots with labels needed. Use rough drafts or wait for final software?

**4.1 Components**

The software architecture of this system is based heavily around modular design. The system is broken into several components, each with a distinct function. The four main blocks are - the user interface, the monitoring system, the main controller and the hardware interfaces.

**4.2 User Input & Interface**

Data flow of this software system, outside of looping states, originates from user input. User input events will trigger certain routines to be executed. The software will process the input and any associated variables/parameters to determine the desired state of multiple hardware components.

Controls for the user interface will be represented in mainly two forms: Boolean switch buttons and analogue dials/open fields for parameter entry.

When a valid user event is triggered, the appropriate command constant is enqueued onto the user command queue to awaiting processing from the main controller.

Display elements of the user interface will also be present to provide feedback information on the system. This will allow monitoring of the status of the system, as well as detailed information for developmental analysis.

**4.3 Monitoring System**

Inputs from the different DAQ pins are repeatedly read within one loop. Each loop takes the current input value and updates a functional global variable. More?

**4.4 Main Controller**

The main controller loop is responsible for receiving user commands via the user command queue, and translating them into parameters that the various blocks further down in the hierarchy can work with.

All of the logical calculations including PID control are centralized in this block.

The output of the main controller is a number of control queues going to their respective control blocks, each carrying the necessary variables to incite an action/inaction.

The controller has multiple states, initialize/running/termination etc

**4.4.1 Pressure (Main controller)**

The most recent pressure reading from the monitoring system is used as the input for this section of code each loop iteration. The input voltage reading is first put through a couple of numeric conversions to give a pressure reading (precalculated, reference excel formulation?). The pressure reading is then fed into a PID control VI alongside a user controlled set point. The control output value (lower limit – 0.4, upper limit 5) is subsequently put onto a queue bound for the pressure hardware interface.

The information pushed onto the queue consists of:

A voltage value that dictates the rate at which the pump operates.

A collection of Boolean values which dictate which valves are opened/closed.

**4.4.2 Temperature (Main controller)**

The monitoring system once again provides the most recently read sensor value as input for the temperature logic. The input voltage reading is converted into a degrees Celsius reading (precalculated, reference excel formulation?) and fed into the PID VI to be compared to the user controlled set point. The output (lower limit – 0, upper limit 100), is first gated by the reinitialize button – delivering a static 0 output to the queue if pressed, control output if not. The control value is then translated into a millisecond value which will dictate the frequency of voltage pulses the heater will receive.

The information pushed onto the queue consists of:

A voltage value that dictates the frequency of voltage pulses the heater will receive.

A Boolean value flag which indicates control output has capped at 0. Heaters are turned off when this flag is raised.

**4.5 Hardware Control Interface**

Elements on the command queues are dequeued by the relevant control block for processing. The common structure is a simple while loop, writing values continuously to a DAQ output. These values are pre-calculated by the main controller and can also vary depending on state machine conditions. There is also an override control present in this functional block, which allows a user to tailor output manually instead of using the PID control output.

Further details of each interface are outlined in the subsections below.

**4.5.1 Pressure**

The state machine within this block consists simply of two states:

The PID control state in which it dequeues the latest control output from the control queue and writes the voltage/Boolean settings to the pump/valves.

A manual override state which uses a user input as the voltage/Boolean setting.

A shutdown state which terminates the loop.

Prime, flush routines etc

**4.5.2 Temperature**

The state machine within this block once again consists of two states:

The PID control state in which it takes a millisecond value and translates it into a corresponding frequency of pulses to the heater. It achieves this by having a time controlled flat sequence structure in which the output is low followed by high.

A manual override state which uses a user input as the heater pulse frequency setting.

A shutdown state which terminates the loop.

Prime, flush routines etc

**4.6 Data Logging**

Provide metrics, flow rate, print rates etc?

**4.7 Termination & Error Handling**

When the user input command for termination has been given, loops are exited in an ordered manner to avoid unexpected hang/crash states. The main controller sends the exit command to all of its output queues, signaling every loop to terminate. Once the main controller has confirmation via a flag (FGV?) that every loop has terminated, it shutdowns itself.

More detail on errors once written

**4.8 Optional Blocks**

**Features not explored yet, agitation etc?**

**5. Architecture Design Decisions**

**5.1 Modular Design**

Earlier design drafts of this software have shown that it is very possible to have much of the code within separate loops/blocks combined. However after careful consideration towards readability, separation of functional blocks seemed a much more sensible approach. The modularity of the software also lowers the difficulty of future development, allowing expansion of features very feasible.

**5.2 Timing Considerations**

Timing in this software is relatively non-rigid in terms of customization, due to the block separation. For example, changing the rate at which sensor samples are taken has little to no impact on the operation of the PID control (outside of providing more recent readings, of course).

Since there is a lot of dependency on information acquired from blocks running at different speeds, communication between functional blocks are designed accordingly so. Lossy enqueue is utilized to avoid queue backlogs of old data, ensuring only the most recent values are acted upon. The monitoring system dealing with sensor readings updates functional global variables that once again, only store the most recent value.

**Anything else missing?**