DYNAMIC MODELING AND HARDWARE IN THE LOOP TESTING OF CHEMICAL PROCESSES

BY

MALIA LAUREL KAWAMURA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Adviser:

Professor Andrew Alleyne

Abstract

This thesis presents a framework for hardware-in-the-loop (HIL) testing of chemical plants. HIL testing is a widespread tool used in industry and academia to bridge the gap between computer simulations and physical experimentation. It provides many advantages to the standard development path of building a physical prototype and then running tests. Benefits of HIL testing include decreased development time, reduced cost, increased safety, and better control algorithm development. For this work, HIL testing consists of an emulated real-time chemical plant and a real physical controller.

This HIL testing setup requires two main thrusts – the development of a dynamic model for chemical plants and the implementation of an emulated real-time plant and a real physical controller in hardware. A user-friendly, modular, scalable, dynamic, and nonlinear first principles modeling toolkit is developed in Matlab Simulink. The toolkit contains individual chemical plant components such as a continuous stirred tank reactor (CSTR), pump, and valve. Experimental validation of the chemical concentration models and an example plant model are included. Next, for the hardware and control implementation tasks, National Instrument's VeriStand software is used to integrate a Simulink model to run in real-time on NI hardware. A myRIO is used as a real physical controller, programmed in LabVIEW, to control the emulated real-time chemical plant running on a CompactRIO. A chemical plant which forms propylene glycol in a CSTR is utilized as a case study. However, the HIL testing framework developed is widely applicable to real physical systems to decrease development time and increase safety.

To my family, friends, and teachers.

Acknowledgements

First, I would like to thank my adviser, Dr. Andrew Alleyne, for his guidance and commitment to helping me develop as a researcher and individual. Your dedication to research and your students' entire experience in graduate school is truly unique. I sincerely appreciate your devotion to creating a collaborative, inspiring, and enjoyable lab environment which shaped my time at University of Illinois.

Second, I would like to thank my family for shaping how I view learning and hard work. My parents have always encouraged me and taught me to believe in myself. My sister, Kiana, pushes me to do my best while simultaneously being an unwavering supporter. Kiana also provided me with the idea to use the crystal violet bleaching reaction.

Third, I would like to thank the people I spent the most time with, my fellow Alleyne Research Group members. Thank you for both improving my research and making my time enjoyable. To the senior lab members, Justin, Matt, Herschel, and Bryan, thank you for being outstanding examples of great resesarchers and your openness to helping with my research. To Pamela, Nate, Spencer, Ashley, and Oyuna, thank you for being excellent classmates and good friends. To Sarah, Sunny, and Spencer (S^3), thank you for bringing such amazing attitudes, talents, and fun to the lab. I really appreciate all of you for making my time both in lab and outside of lab so enjoyable.

Finally, I would like to acknowledge the financial support provided by the Dow Chemical Company, the National Science Foundation (NSF), and the Graduate Research Fellowship Program (GRFP). This support has given me the flexibility to conduct research which interests me and has societal value.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This thesis investigates how dynamic modeling can be paired with real-time testing in hardware to better understand system behavior, develop plant designs, and implement control algorithms. The methodology of modeling and implementation of models in real-time on hardware could be applied to many different energy systems. In general, models allow plant design and control algorithm decisions to be quickly tested in hardware rather than necessitating physical plants early in the design phase. Reducing the need for physical plants in the design process saves time, money, and energy as shown in Figure 1.1. The iteration process with physical plants can be slower than the iteration process with models. This thesis focuses specifically on chemical plants as a case study. Chemical plants are studied because there is a need for better dynamic modeling to capture transient characteristics of the chemical processes. Additionally, more detailed models could allow more complex control techniques to be investigated because current control algorithms are typically based on steady state behavior rather than dynamic behavior [1].

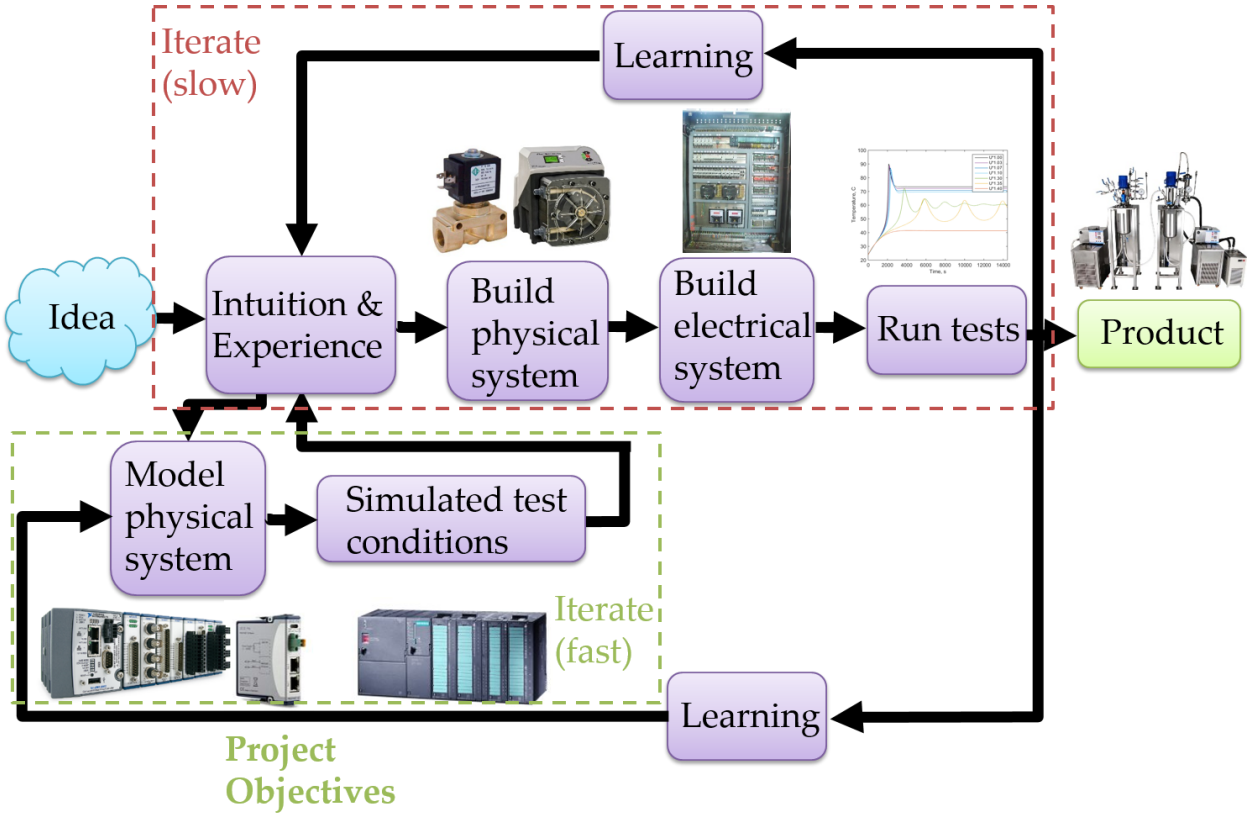

Figure 1.1 A schematic depicting two alternative development and testing paths for new products highlighting the shorter development path that this thesis will investigate.

1.1 Motivation

The unsustainable dependence of the United States on fossil fuels is highlighted by the increase in adverse effects from their use: rising global temperatures, increasing sea level, and new, dangerous, weather patterns [2]. The Clean Power Plan includes a piece of the solution to fossil fuel dependence by requiring 30% more renewable energy generation by 2030 [2]. However, increasing the use of renewable technologies, such as solar and wind energy, is only one aspect of sustainable energy generation. Conserving energy by considering entire systems and optimizing their efficiency is an equally important and complementary avenue to reach energy sustainability. According to the U.S. Energy Information Administration, chemical industries consume 29 percent of the total energy consumed in U.S. manufacturing in 2002 [3]. Therefore, in the interest of conserving energy, both the chemical industry and the United States are invested in examining chemical industry energy use.

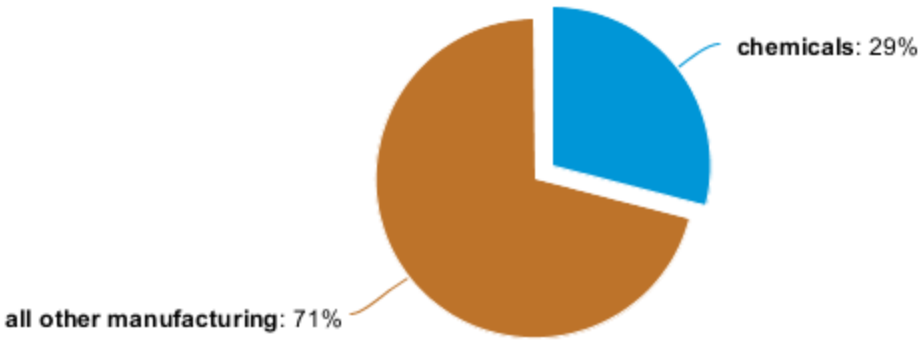

Figure 1.2 Chemical industries used 29 percent of the total energy consumed in U.S. manufacturing in 2002 [3].

1.1.1 Information Driven Energy Systems

With the rapid development of many new energy related technologies, there is a need for a quicker way to test novel ideas and system designs without building large physical plants. For instance, solar farms, wind farms, chemical plants, vehicle platforms, and other complex machines are expensive, energy consuming, and slow to build. Ideally, these information driven energy systems could be modeled, simulated, and optimized using hardware-in-the-loop (HIL) testing prior to full-scale construction.

1.1.2 Introduction to Hardware-in-the-Loop (HIL)

The term hardware-in-the-loop (HIL or HWIL) can have different meanings, but in general it refers to a plant and controller connected in a loop together with at least one part in simulation and at least one physical part. In other words, either there is a physical plant or a physical hardware controller as shown in Figure 1.3. More formally, "Hardware-in-the-loop simulation (HIL) is characterized by the operation of real components in connection with real-time simulation components" [4]. An important aspect of HIL testing is that it must be done in real-time. The highlighted form in Figure 1.3 with the emulated plant and real physical controller is the HIL configuration implemented in this thesis.

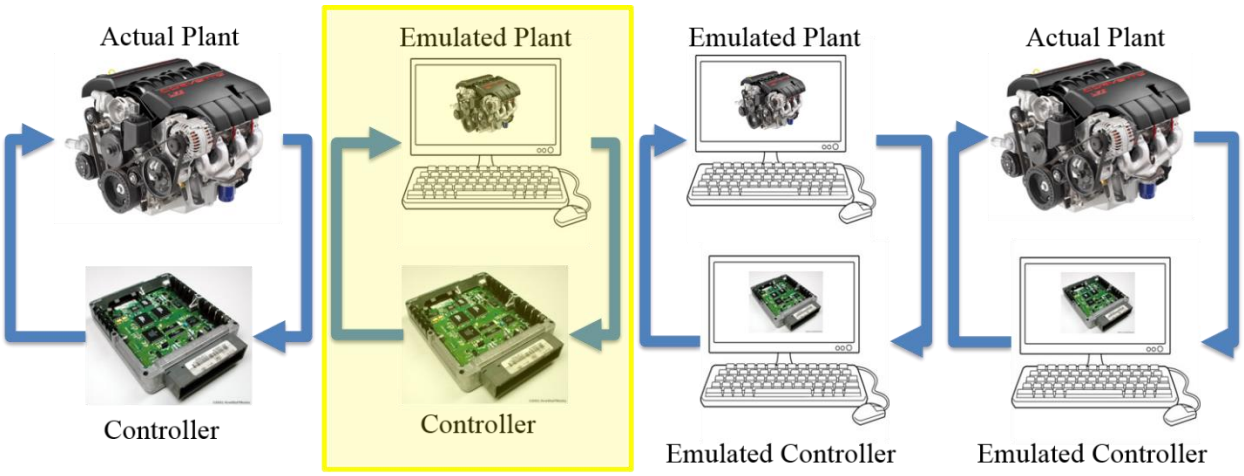

Figure 1.3 Different examples of HIL configurations showing the generality of the term HIL as well as highlighting the form of HIL that will be implemented in this thesis.

1.1.3 Historical Development of Hardware-in-the-Loop (HIL)

In the past, HIL has sometimes been categorized more strictly as shown in Figure 1.4 with HIL only referring to simulated processes and real control systems. However, currently, the definition of HIL is looser to encompass a real process and simulated control system as well as shown in Figure 1.3.

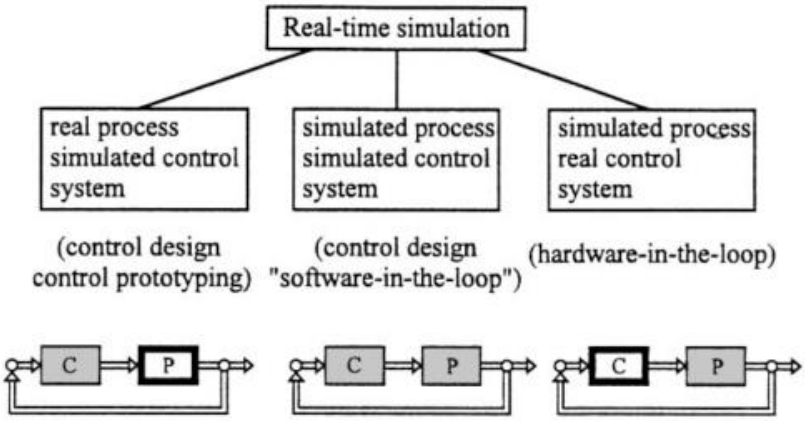

Figure 1.4 Real-time simulation classifications showing what was considered hardware-in-the-loop in 1999 [4].

The original application for HIL testing was likely to train pilots with flight simulations in 1936 [4]. For pilot training, real pilots interacted with a real cockpit that had real actuators

with a simulated process and simulated sensors. For this situation, the advantages of training pilots using this HIL setup included the ability to test extreme and dangerous operating conditions, the saving of money and development time, and the capability of reproducible experiments [4].

1.1.4 Current Industrial Use of Hardware-in-the-Loop (HIL)

Since 1936, many other industries beyond the aerospace industry have proceeded to develop new processes and products using HIL simulation or HIL testing. HIL is a key step along the rapid development path to bridge the gap between ordinary simulations and experiments on real plants [5]. HIL is particularly common in the automotive industry and has been used for modeling engine control systems [4], engine fuel systems [6], and electric vehicle functional torque safety [7]. It is notable that in the ASME Dynamic Systems and Control Division (DSCD) newsletter, which features works which are of broad interest to the community, the Winter 2016 article focused on the HIL implementation for fault diagnosis for torque functional safety of electric vehicles [7]. Additionally, HIL testing has been used in the transportation industry for rail vehicle control [8] and traffic engineers developing signal timing plans [9]. The chemical industry relies on HIL testing as well for a wide variety of applications including solar cooling in fermentation units [10] and distillation columns [11]. Therefore, the automotive, aeronautical, transportation, and chemical industries are a few examples of where HIL testing is already used both in industry and academic research.

1.1.5 General Benefits of Hardware-in-the-Loop (HIL)

Perhaps most importantly, the numerous benefits of HIL transfer across different industries. Cost and time savings are two key benefits of HIL testing. Specifically, HIL simulation saves time and money when the dynamics of a chemical process can be modeled with good precision and fidelity and when experiments with the real process would create an unnecessary cost because of the time needed to install and test the hardware [12]. Other chemical process papers cite decreased development time as an advantage of HIL in general and especially when physical components are unavailable and saved time directly translates to saved cost [10], [13]. In the case of traffic engineers, the ability to use HIL testing for developing signal timing

plans saves both engineers and civilians time by reducing the need for real street testing which greatly reduces minor and major traffic disruptions [9].

Another key benefit of HIL testing that transcends different industries is increased safety. Returning to the application of traffic engineers developing signal timing control, it is clear how testing plans in the security of an office without endangering human drivers increases safety [9]. Additionally, HIL in the chemical process industry has been acknowledged as a method to avoid unnecessary risks in projects that can compromise the safety of people [12]. Specifically, allowing process operators to learn the proper response to various conditions before experiencing the conditions on an actual process increases safety for all employees [14]. The safety advantages of HIL testing become extremely important for processes that achieve optimal performance near physical and operating constraints [15].

Increased accuracy and control algorithm development are additional benefits of HIL testing. Specifically, having a controller or real component in hardware enables the avoidance of model mismatches and thereby increases the reliability of simulation results [13]. For chemical processes, it is acknowledged that a HIL testing method improves the accuracy of experiments whenever one of the components involved in simulation is physically available [10]. As for control algorithm development, it provides engineers time to develop, model, and test control schemes, such as signal timing plans, and there is less need to fine-tune control under actual conditions [9]. For vehicle development, HIL testing is a part of the development path for developing control and it is acknowledged that "HIL testing is an important step to test the effectiveness of algorithms before they can be implemented in a real vehicle" [7].

One of the powerful aspects of HIL testing is that it operates in real-time. Developing the best control algorithms and testing the control in real-time is crucial because timing is very important for real processes [12]. For example, timing is key for model predictive control (MPC). HIL allows the testing of MPC as the control method in real-time to see whether the computations can be calculated fast enough to avoid any errors and result in the desired process behavior. Further discussion of how HIL development is useful in the chemical process industry is included in Chapter 5.

1.2 Thesis Outline

This thesis is organized as follows. Chapter 2 introduces how chemical plants and processes operate currently. Then, the dynamic chemical plant modeling toolkit developed in Matlab Simulink is explored in detail. Chapter 3 explains how the model components in Chapter 2 were validated. Specifics about the chemical reaction concentration model validation with an experimental setup are included. Chapter 4 provides a representative chemical plant model as an example of how the modeling toolkit captures dynamic plant behavior. Chapter 5 develops why HIL is beneficial for the chemical process industry. Additionally, it provides specifics about how the Simulink plant model developed in Chapter 4 is run on hardware in real-time. Chapter 6 provides a demonstration of the HIL implementation with a simple control algorithm on the example chemical plant model. The thesis concludes in Chapter 7 with a summary of research contributions and direction for future work.

Chapter 2
Chemical Process Modeling

2.1 Introduction to Chemical Plants and Processes

There is a wide variety in chemical plants including what the plant produces or processes, the physical plant size, and the relative rates of reaction in all of the subsystem processes. Examples of chemical plants include water treatment plants, coal burning electrical power stations, and new chemical material production [16]. Industrial chemical plants vary in size from less than a gallon to holding over 400 tons of material [17]. For example, kilns that produce lime may be over 25 meters high as shown in Figure 2.1. The variety in sizes can be attributed in part to advantages of specific reactor types such as a batch reactor, continuous stirred tank reactor (CSTR), plug flow reactor, etc. This thesis will focus primarily on CSTRs, which in particular need to be large reactors to obtain high rates of conversion [18]. The variety of time scales for chemical reactions is also diverse. Producing polyethylene, one of the most important plastics, or gasoline from crude petroleum have reaction steps on the order of seconds in contrast to waste water treatment which can have a reaction that takes days [16]. To represent the relationships between different common components, such as pumps, pipes, valves, and reactors, it is common to use piping and instrumentation diagrams or drawings (P&IDs). Figure 2.2 is an example of a P&ID for a process which includes a chemical reactor, a flash separator, and heat exchangers [19].

Figure 2.1 Lime kiln in Wyoming showing the large physical scale of many chemical industrial processes [20].

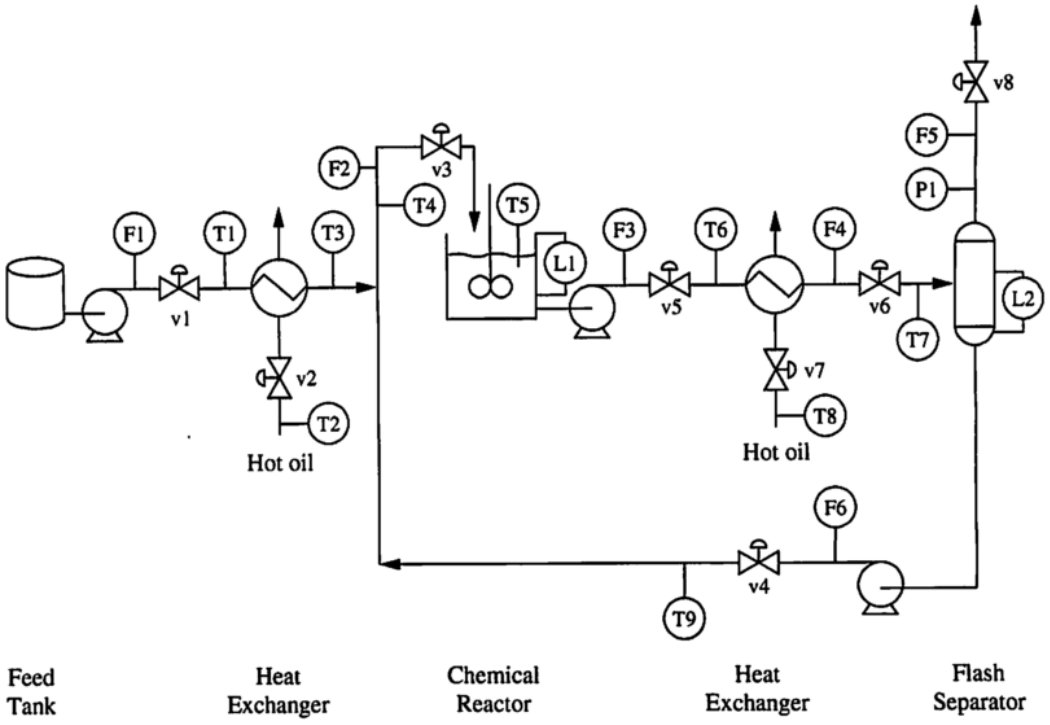

Figure 2.2 Example piping and instrumentation diagram (P&ID) to show how chemical plants are typically represented schematically [19]**.**

2.1.1 Current Modeling Practices

For chemical process modeling, for both plants and components, there are two primary modeling software packages utilized in industry. Aspentech's Aspen HYSYS is the leading process simulation software used for process optimization in design and operations [21]. Aspen HYSYS is very commonly used in industry, including the world's leading oil and gas organizations [12], [22], [23]. This modeling environment allows for individual components, such as a distillation column, or entire plants, such as a gas plant, to be modeled [21]. Despite this being a common industry modeling software, cost remains prohibitive for exploration in an academic setting and for some companies. Mathworks' Matlab Simulink is another popular modeling choice [5], [24]–[26]. Matlab Simulink provides a block diagram environment that allows for object oriented dynamic system simulation [24], [27]. Simulink allows users to build custom libraries to create dynamic models of entire plants as well as investigate single components. An advantage of Simulink is that the low-cost education license makes it accessible for academics [24]. Of course other modeling options exist, but Aspen HYSYS and Matlab Simulink are the two most commonly used modeling software options.

To create reasonable and functional models of plant wide industrial processes, there has been a need for a realistic physical scenario. Downs and Vogel's paper, A Plant-Wide Industrial Process Control Problem, published in 1993 describes a model of an industrial chemical process and has since been cited over 1450 times [28]. This chemical process problem is often referred to as the Tennessee Eastman test problem and is unique in its physical grounding involving two simultaneous gas-liquid exothermic reactions. The rarity of information and data for realistic plants and chemical reactions in the open literature continues to be a modeling challenge.

Modeling plant wide processes is useful for testing different control approaches. Specifically, some typical chemical process control objectives include maintaining process variables at desired values, such as a desired steady state temperature [28]. Additionally, it is often necessary to keep process operating conditions within equipment constraints, such as an obtainable mass flow rate through a pump [28]. Other objectives include minimizing variability of product rates, minimizing movement of valves, and recovering quickly and smoothly from disturbances [28]. Additionally, control objectives can be formulated around minimizing cost [1].

Therefore, component and plant models have value for better understanding process dynamics and designing control approaches in simulation. However, there is a disconnect between pure modeling and simulation and applying the control techniques designed in real hardware on real physical plants. Thus, it is important to examine the hardware used in the process control industry.

2.1.2 Current Chemical Plant Hardware

A programmable logic controller (PLC) is an industrial digital computer and an industry standard for the control of manufacturing processes [24], [29]. PLCs are the primary hardware used for control because they have been ruggedized to be reliable and robust systems. Additionally, the ease of programming and debugging, the dedicated I/O, and the ability for memory expansion explain why PLCs are the industry standard [29]. Of course, there are some shortcomings of PLCs, which become especially influential when more complicated control schemes need to be implemented. Specifically, matrix operations such as transposition, inversion, and multiplication are difficult to perform [29]. Since PLCs work in real-time, which means they need to provide a response within the specified time constraints, computations need to be done within the sampling period. The effect of this time constraint is that the computational load needs to be kept as low as possible to avoid error accumulation [29]. Furthermore, even though an advantage of PLCs is that memory can be extended, it is also a hindrance since it means that larger programs which require more memory have an associated cost tradeoff [29]. The benefits and constraints of PLCs, the industry standard hardware, need to be incorporated while considering control algorithm options.

Another popular hardware option in the process control industry are dSPACE systems [30]. The German company, GmbH, produces dSPACE systems which are fast real-time hardware systems. dSPACE systems outperform PLCs for high-speed control and data analysis [24]. It is important to note that dSPACE systems are also widely used for rapid prototyping and HIL simulation in many application areas such as the automotive industry [7], [8], the aerospace industry [4], and electronics industry [31].

2.1.3 Current Chemical Plant Control Methods

Control techniques in the chemical process industry are typically based on steady state behavior or optimal process set-points rather than dynamic behavior [1]. The most widely used control algorithm in the chemical industry is proportional-integral-derivative (PID) control because it is well understood by control engineers [5], [6]. Specifically, regulatory control layers tend to include mostly single-input single-output (SISO) control loops such as PID control loops [1]. Likewise, any lower level control system in industrial processes is usually equipped with linear controllers with PID action [15]. Specific examples of typical uses of PID controllers in industry include pH, temperature, and liquid level control loops [32]. Additionally, the PID regulatory goals align well with the steady state control paradigm frequently found in the process control industry.

The other main type of control implemented is Model Predictive Control (MPC), which is a more advanced method of process control than PID. MPC uses a dynamic model of the process to predict the future response of the system. MPC makes control decisions which minimize a specified cost function over the prediction horizon by solving a finite-time, open-loop optimal control problem. Interestingly, MPC was developed in industry for applications which required the operation of systems near their limits to improve production beyond the capability of PID controllers [33]. Now, MPC has widespread use in the petrochemical and chemical sectors of the process industry in large plants, but is not as widely used in medium and small plants [12]. It is important to note that MPC requires an optimization problem to be solved in real-time which usually involves significant on-line computation. MPC has been attempted on industrial PLCs and it has been shown to be possible to have sampling times on the order of milliseconds [12]. Nevertheless, the number of constraints and control variables with MPC are restricted due to computational hardware limitations [12]. It should be noted that MPC is often applied in a supervisory control layer rather than in the lower layers which more commonly utilize PID control [1].

A variation of MPC is economic MPC (EMPC), which integrates economic process optimization and process control. EMPC uses a cost function which can be directly or indirectly related to the process economics. An advantage of EMPC is that it incorporates cost and allows

for the optimization of processes in a dynamic fashion by not operating processes at a specified steady state target [1].

In practice, since chemical plants are physically large, control is usually implemented in a central control room. In the control room, an individual can be responsible for monitoring up to 100 controlled variables and 400 other measured variables [19]. There are other individuals who monitor the complex plant directly with local displays and can make local changes as shown in Figure 2.3.

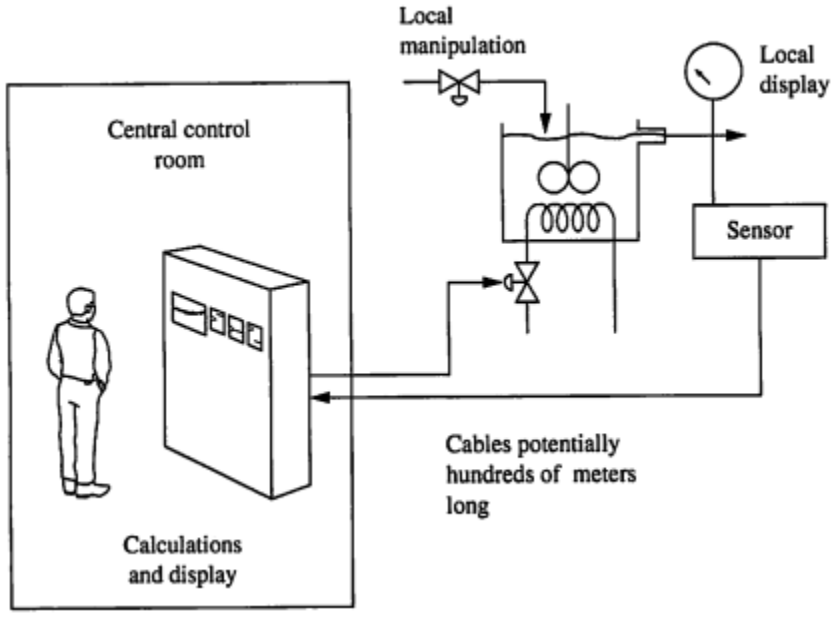

Figure 2.3 Schematic showing how a typical control system has both local and centralized control [19]**.**

2.2 Modeling Features

This section describes the main features of the modeling toolkit developed in Matlab Simulink which is well suited to modeling the dynamics of chemical processes a in physically meaningful way.

2.2.1 Modularity

In Simulink, a library of chemical plant components was created. The library consists of separate blocks that each represent a different plant component such as a CSTR, CSTR jacket,

pump, valve, and pipe shown in Figure 2.4. The modularity of these components is similar to how the chemical industry currently represents plants using P&IDs and the icons for each component match the P&ID icons. Each physical component being self-contained in a block allows the user extreme flexibility in constructing a plant structure. Additionally, the modularity makes it very easy to edit particular components and add new blocks.

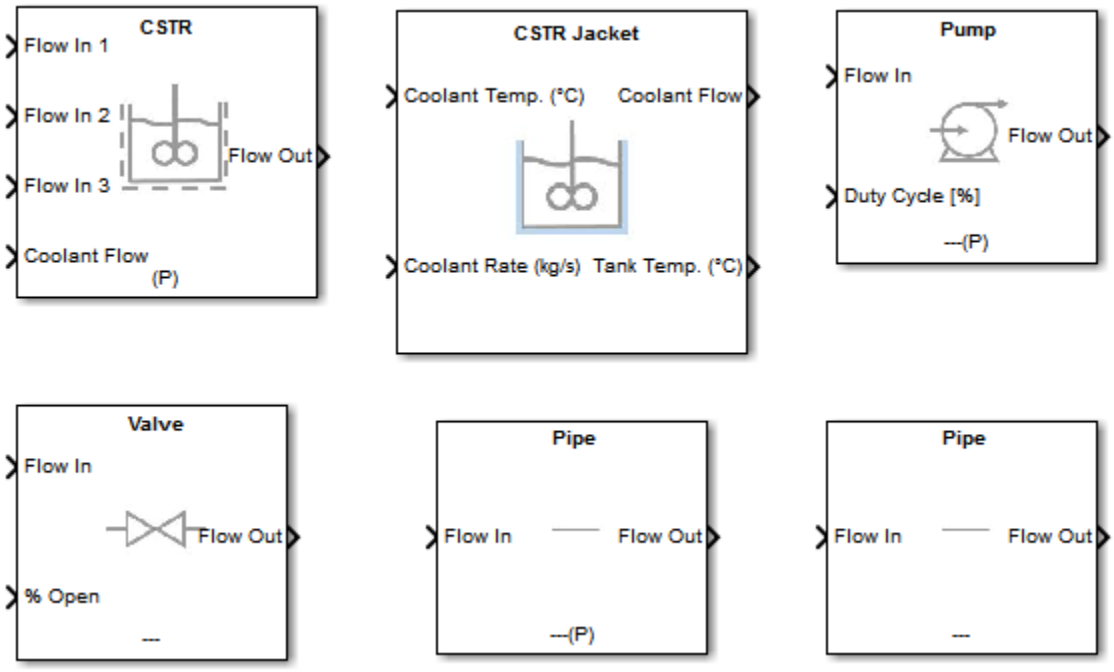

Figure 2.4 Main chemical process library components.

2.2.2 Dynamic Modeling

Each modeling block tracks the mass flow rate and pressure. The modeling framework is constructed to allow the pressure and mass flow rate of blocks upstream and downstream of the current block to influence the current block's dynamics. The interaction of neighboring blocks and what values are needed to calculate the mass flow rate or pressure (since they are related) requires the blocks to be connected properly. To ensure the correct values are available between neighboring blocks, each block has a small pictorial representation in the bottom center as shown on the pump block in Figure 2.5. The schematic meanings are explained in Table 2.1. The key to connecting the blocks is to always have the mass flow rate (dashes) and pressure (P) symbols

alternate. In other words, there would not be two blocks connected like --- and ---(P) such as a valve --- and pump ---(P) because that means two mass flow rates are connected directly.

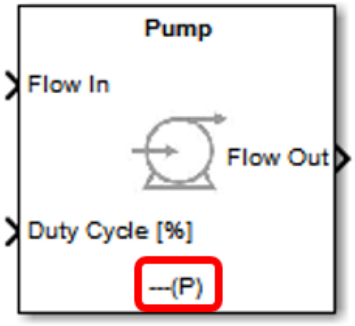

Figure 2.5 Pump block demonstrating the mass flow rate and pressure symbol at the bottom center.

Table 2.1 Schematic symbol and explanation for what each library block calculates.

Symbol	Block calculates	Receive from upstream block	Receive from downstream block	Send to upstream block
---	Dynamic mass flow rate	Pressure (inlet)	Pressure (outlet)	Mass flow rate (inlet)
(P)	Dynamic pressure	Mass flow rate (inlet)	Mass flow rate (outlet)	Pressure (inlet)
---(P)	Dynamic outlet pressure and algebraic inlet mass flow rate	Pressure (inlet)	Mass flow rate (outlet)	Mass flow rate (inlet)

The CSTR block calculates additional dynamic states beyond mass flow rate and pressure. In particularly, the CSTR dynamically models the liquid height, temperature, and concentrations of six reagents. In the future, the CSTR model could be modified to calculate the concentrations of additional reagents, if desired, but most examples do not have more than three reactants and three products.

2.2.3 Fluid Property Tables

For particular reactions, it could be useful to integrate relevant property tables to dynamically solve chemical reactions. This was not found to be needed directly for any reactions

tested, but the framework for incorporating this capability is included. Specifically, for water, there is a two dimensional lookup table for the density of water based on the temperature and pressure. In the future, these lookup tables could be added for other fluids as needed.

2.2.4 Physically Meaningful Parameters

A feature of this modeling toolkit is that each component allows the user to enter physically meaningful parameters. Other educational chemical modeling programs often have users enter only a few parameters, which can obscure the physical meaning. For example, with Reactor Lab, as shown in Figure 2.6, a user enters a single value for the product of the heat transfer coefficient (U) and the surface area (A) [34]. In contrast, Figure 2.7 shows how for the newly developed modeling toolkit, the user enters the reactor diameter and height which are physically meaningful values as well as a separate heat transfer coefficient value. PISim, another educational modeling tool, has similar shortcomings with few physically realistic parameters available for the user to adjust [35].

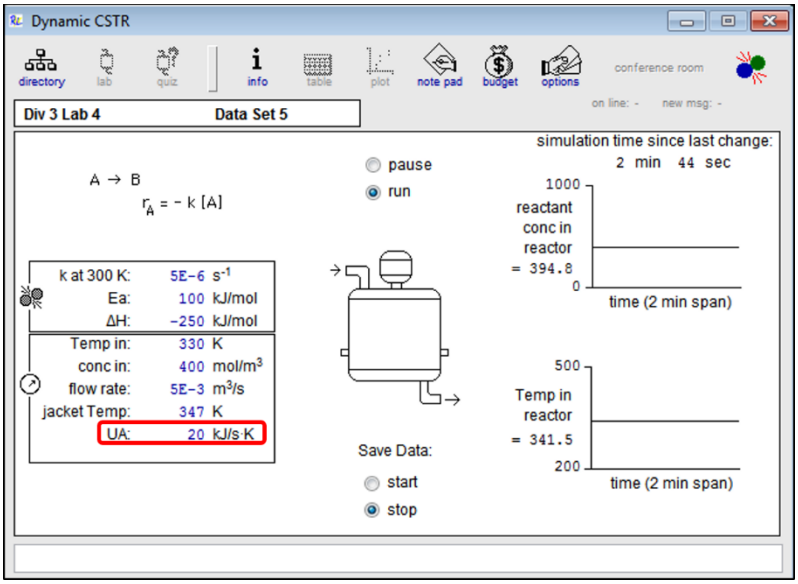

Figure 2.6 Reactor Lab CSTR user interface showing one parameter to enter for the product of the heat transfer coefficient and surface area [34]**.**

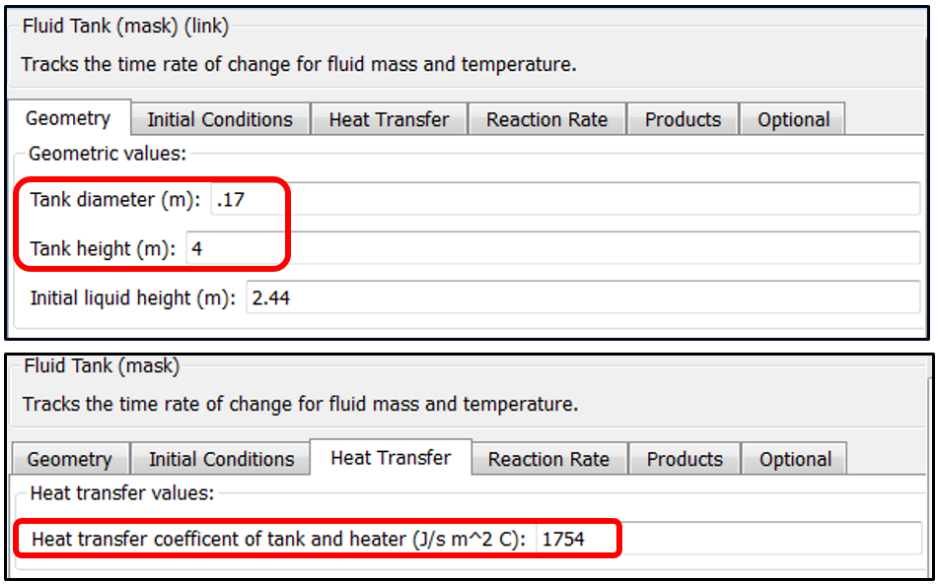

Figure 2.7 CSTR block user interface options showing the geometry parameters in one tab and the heat transfer coefficient in another tab to maximize physical meaning.

2.2.5 Bus Structure

In a model, each block is connected by a "fluid flow" signal. Examples of this "fluid flow" are provided in Figure 2.8. This "fluid flow" is a bus structure that contains seven values including temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation for the fluid as shown in Figure 2.9. The grouped three lines represent a bus signal. The integration of all these values grouped together allows the models to require fewer connections when constructing a plant model and the bus structure allows signals to be selected by name.

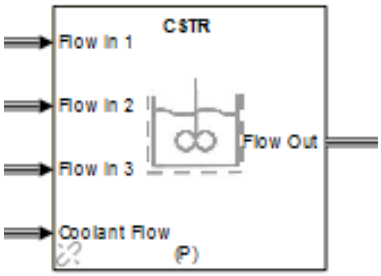

Figure 2.8 CSTR block showing the three inlet fluid flows and one outlet fluid show demonstrating the built in bus structure of the modeling toolkit.

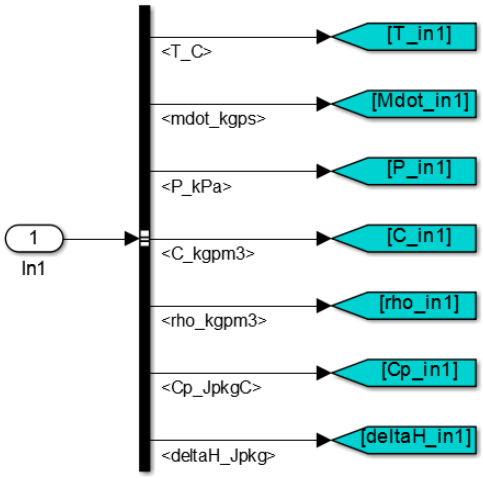

Figure 2.9 Example of the bus structure utilized in the modeling toolkit showing how a "fluid flow" that enters as a single flow in the inport, In1, contains seven values.

2.2.6 Summary of Modeling Toolkit Advantages

A key advantage of this modeling toolkit is the ease of use. The physically meaningful parameters, schematic at the bottom of each block to show users what order to connect blocks in, and the bus structure all contribute to making the toolkit user friendly. Additionally, the modularity to allow changes to specific components and the ability to add new components makes this tool scalable. Finally, the use of only native building blocks in Simulink means that users do not need any extra toolkits to run the model.

2.3 Modeling Toolkit Block Specifics

The modeling toolkit contains ten main components and four support functions:

- ☐ CSTR
- ☐ CSTR Jacket
- ☐ Pump
- ☐ Standard Pipe
- ☐ Secondary Pipe
- ☐ Valve
- ☐ Mass Flow Rate Source
- ☐ Pressure Source
- ☐ Mass Flow Rate Sink
- ☐ Pressure Sink
- ☐ Data Manager (support function)
- ☐ Data Sink (support function)
- ☐ Sink (support function)
- ☐ Source (support function)

All of the Matlab code for the modeling toolkit is included in Appendix A.

2.3.1 Continuous Stirred Tank Reactor (CSTR)

The continuous stirred tank reactor is where the chemical reaction occurs. A CSTR has continuous flow in and out. The continuous flow in and out is what differentiates a CSTR from a batch reactor, which has no flow in or out (besides the initial flow of reactants in). CSTRs have stirring mechanisms to keep the products well mixed. Additionally, most CSTRs have a cooling or heating jacket on the outside which can serve to cool or heat the substance inside the reactor. The CSTR jacket is depicted in blue in Figure 2.10.

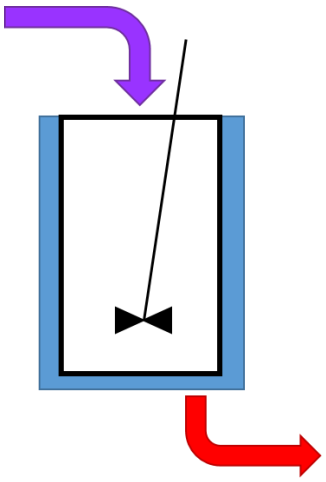

Figure 2.10 Schematic of a CSTR showing a continuous flow in and out of the reactor, a cooling jacket, and stir rod.

2.3.1.1 Mathematical Model

For the CSTR, the energy balance is

$$\frac{dT}{dt} = \frac{\dot{Q} - \dot{W} - F_{A0} c_{ps} (T - T_0) + \Delta H_{RX} (r_A V)}{C_{A0} V c_{ps}}, \tag{2.1}$$

where T is the temperature in the reactor and other parameters can be found in Table 2.2 [36]. For simplicity, the temperature is assumed to be uniform through the reactor [36]. If the tank is heavily stratified, then the chemical reaction can be broken up to occur in two CSTRs with flow between them. In the numerator, the first term, $\dot{Q}$, represents the heat transfer from the jacket to the reactor. The second term, $\dot{W}$, represents the shaft work which is assumed to be zero. The third term represents the heat transfer between the inlet flows and fluid in the reactor. The fourth term represents the heat generated by the chemical reaction itself. The following equations assume a reaction mechanism of the form

$$aA + bB + cC \rightarrow dD + eE + fF, \tag{2.2}$$

where the lower case letters represent a numeric coefficient and the upper case letters represent a chemical species. For example, with the reaction

$$CH_4 + 2O_2 \rightarrow CO_2 + 2H_2O, \tag{2.3}$$

the matching means that $a=1$ and $A=CH_4$ or methane, $b=2$ and $B=O_2$ or oxygen, $c=0$, $d=1$ and $D=CO_2$ or carbon dioxide, $e=2$ and $E=H_2O$, and $f=0$. Equation (2.2) is a standard representation for chemical reactions [36], [37].

The heat transfer from the jacket exchanger to the reactor is usually simplified to be given by

$$\dot{Q}=UA_r(T_a-T), \tag{2.4}$$

where T_a is the jacket temperature and U is the heat transfer coefficient [37]. The surface area of the reactor is calculated by

$$A_r=2\pi(\frac{d}{2})h_t, \tag{2.5}$$

which assumes the contact area between the jacket and tank is around the outside of a cylindrical tank, but not on the base or top.

The heat capacity of the solution is given by

$$c_{ps}=\theta_A c_{p,A}+\theta_B c_{p,B}+\theta_C c_{p,C}-\theta_D c_{p,\mathrm{D}}-\theta_E c_{p,E}-\theta_F c_{p,F}, \tag{2.6}$$

where $c_{\mathrm{i},A}$ is the heat capacity of species i and θ_i is given by

$$\theta_A=\frac{F_{A0}}{F_{A0}};\ \theta_B=\frac{F_{B0}}{F_{A0}};\ \theta_C=\frac{F_{C0}}{F_{A0}};\ \theta_D=\frac{F_{D0}}{F_{A0}};\ \theta_E=\frac{F_{E0}}{F_{A0}};\ \theta_F=\frac{F_{F0}}{F_{A0}}, \tag{2.7}$$

where F_{i0} is the inlet mass flow rate of species i [18].

The inlet temperature is calculated by

$$T=\frac{F_{A0}T_{A0}+F_{B0}T_{B0}+F_{C0}T_{C0}}{F_{A0}+F_{B0}+F_{C0}}, \tag{2.8}$$

which averages the temperature of the inlet mass flow rates.

The heat of reaction is calculated with

$$\Delta H_{RX}=\Delta H_R^{\circ}(T_R)+\Delta c_p(T-T_R), \tag{2.9}$$

$$\Delta H_R^\circ(T_R) = \frac{f}{a} H_F^\circ(T_R) + \frac{e}{a} H_E^\circ(T_R) + \frac{d}{a} H_D^\circ(T_R) - \frac{c}{a} H_C^\circ(T_R) - \frac{b}{a} H_B^\circ(T_R) - H_A^\circ(T_R), \quad (2.10)$$

$$\Delta c_p = \frac{f}{a} c_{p,\mathrm{F}} + \frac{e}{a} c_{p,\mathrm{E}} + \frac{d}{a} c_{p,\mathrm{D}} - \frac{c}{a} c_{p,\mathrm{C}} - \frac{b}{a} c_{p,\mathrm{B}} - c_{p,\mathrm{A}}, \quad (2.11)$$

where T_R is a reference temperature [18].

The rate law is given by either

$$r_A = -k C_A^{\,a} C_B^{\,b} C_C^{\,c}, \quad (2.12)$$

or

$$r_A = -k C_A^{\,\alpha} C_B^{\,\beta} C_C^{\,\gamma}, \quad (2.13)$$

where Equation (2.12) is the elementary rate law form based on the reaction mechanism given in Equation (2.2) and Equation (2.13) is the non-elementary rate law form based on user inputted values α, β, and γ [18]. The rate constant, k, is given by either

$$k = A e^{-\frac{E}{RT}}, \quad (2.14)$$

or

$$k = A e^{-\frac{E}{R}\left(\frac{1}{T_k} - \frac{1}{T}\right)}, \quad (2.15)$$

where Equation (2.14) is the Arrhenius equation and Equation (2.15) is an alternative rate constant equation that is sometimes used [18].

The volume term found in energy balance, Equation (2.1), refers to the liquid volume in the reactor. A cylindrical reactor is assumed so that

$$V = \pi\left(\frac{d^2}{4}\right)h, \quad (2.16)$$

where h is the liquid height in the tank. The liquid height is calculated dynamically using the conservation of mass equation

$$\frac{dh}{dt} = \frac{\dot{m}_{in} - \dot{m}_{out}}{\rho A_c}, \quad (2.17)$$

where $\dot{m}_{in}$ is the total mass flow rate in, $\dot{m}_{out}$ is the total mass flow rate out, $A_c = \pi(\frac{d^2}{4})$ is the reactor cross sectional area and the fluid density is averaged with

$$\rho = \frac{F_{A0} + F_{B0} + F_{C0}}{\frac{F_{A0}}{\rho_A} + \frac{F_{B0}}{\rho_B} + \frac{F_{C0}}{\rho_C}}. \tag{2.18}$$

For the CSTR, the mass conservation concentration equations for the six reagents are given by

$$\frac{dC_A}{dt} = \frac{C_{A0}v_0}{V} - \frac{C_A v_0}{V} + r_A, \tag{2.19}$$

$$\frac{dC_B}{dt} = \frac{C_{B0}v_0}{V} - \frac{C_B v_0}{V} + r_B, \tag{2.20}$$

$$\frac{dC_C}{dt} = \frac{C_{C0}v_0}{V} - \frac{C_C v_0}{V} + r_C, \tag{2.21}$$

$$\frac{dC_D}{dt} = \frac{C_{D0}v_0}{V} - \frac{C_D v_0}{V} + r_D, \tag{2.22}$$

$$\frac{dC_E}{dt} = \frac{C_{E0}v_0}{V} - \frac{C_E v_0}{V} + r_E, \tag{2.23}$$

$$\frac{dC_F}{dt} = \frac{C_{F0}v_0}{V} - \frac{C_F v_0}{V} + r_F, \tag{2.24}$$

where the first term captures the amount flowing into the reactor, the second term captures the amount of flowing out of the reactor, and the third term captures the generation or consumption of the reagent due to the chemical process [18]. The reaction rates are given by

$$\frac{-r_A}{a} = \frac{-r_B}{b} = \frac{-r_C}{c} = \frac{-r_D}{d} = \frac{-r_E}{e} = \frac{-r_F}{f}, \tag{2.25}$$

in relation to r_A [18]. However, if Equation (2.25) does not hold, there is the option to use a scaling factor such as

$$r_B = scale_B r_A. \tag{2.26}$$

Table 2.2 CSTR parameter and state definitions.

A	pre-exponential factor	k	rate constant
A_t	tank surface area	$\dot{m}_{in}$	total inlet mass flow rate
C_A	reactant A concentration	ρ	fluid density
C_{A0}	inlet concentration of reactant A	r_A	rate law of reactant A
$c_{p,A}$	heat capacity of reactant A	R	gas constant
Δc_p	overall change in heat capacity	T	temperature
c_{ps}	solution heat capacity	T_a	jacket temperature
d	tank diameter	T_{A0}	inlet temperature of reactant A
E	activation energy	T_0	inlet temperature
F_{A0}	inlet mass flow rate of reactant A	T_k	rate law temperature
h	liquid height in tank	T_R	reference temperature
h_t	tank height	U	jacket heat transfer coefficient
ΔH_{RX}	heat of reaction	V	tank liquid volume
$\Delta H^\circ_R(T_R)$	heat of reaction at T_R	v_0	inlet total volumetric flow rate
$\Delta H^\circ_A(T_R)$	heat of reaction of reactant A at T_R	$\dot{W}$	shaft work

2.3.1.2 User Inputs

The CSTR block has values for the user to enter in a GUI that opens when the block is clicked on. There are six tabs organized by parameter type. First, in Figure 2.11, the geometry tab has the user enter the tank diameter, tank height, and initial liquid height in the tank. The tab automatically generates geometric values that may be of interest to the user such as the tank volume, initial liquid volume, and tank surface area since these values are used in the energy and mass balance equations.

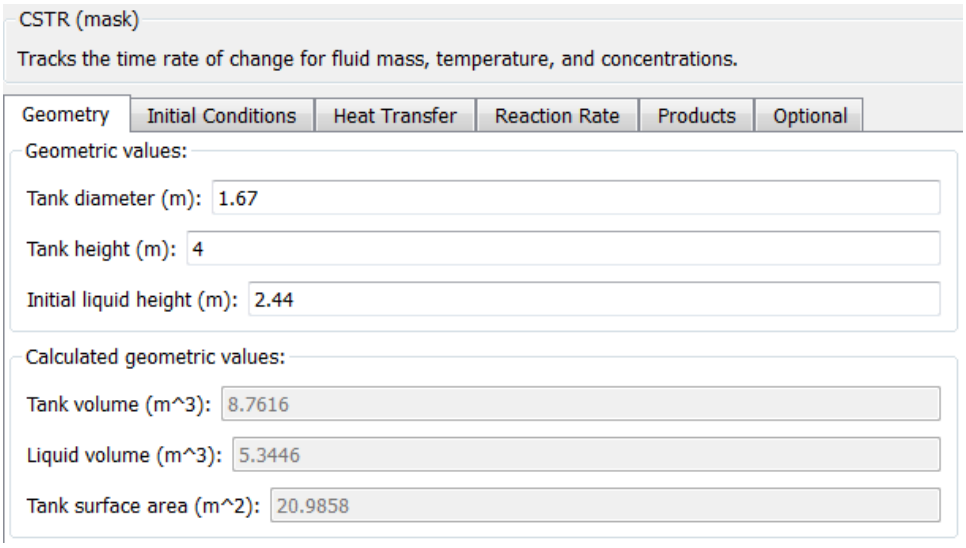

Figure 2.11 CSTR geometry parameters tab.

Next, Figure 2.12 shows the initial conditions tab in which the user enters the initial fluid temperature, initial concentration of the possible three reactants and the pressure inside the tank at the inlet. If there are not three reactants, then the user can enter any values for the unused initial reactant concentrations, but the space cannot be left empty.

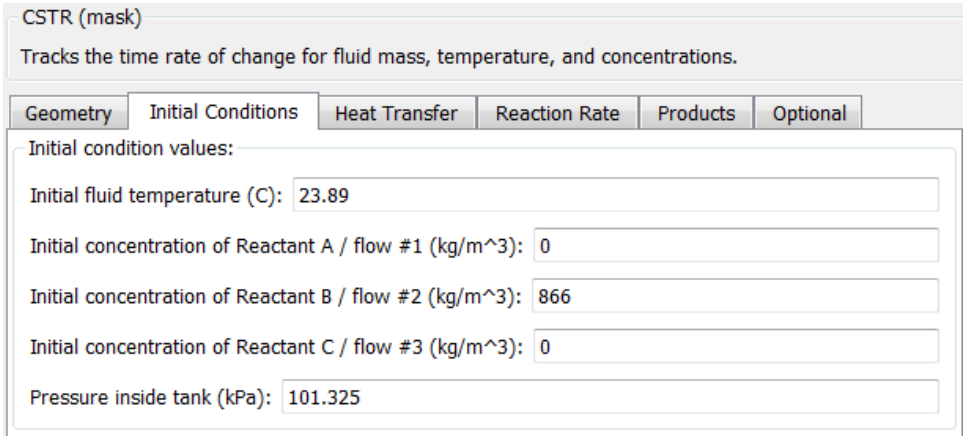

Figure 2.12 CSTR initial conditions parameters tab.

Figure 2.13 shows the heat transfer tab which only requires the heat transfer coefficient between the tank and heater jacket, which is used in the energy balance, Equation (2.1).

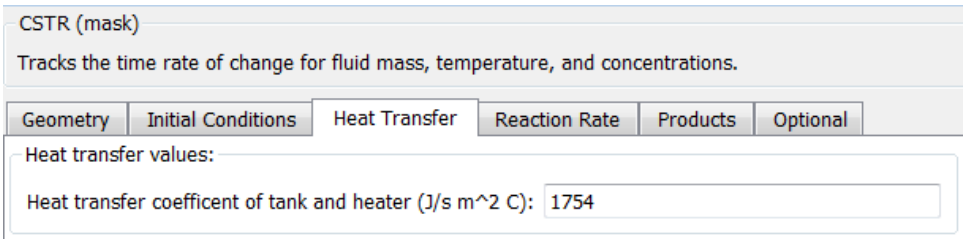

Figure 2.13 CSTR heat transfer parameters tab.

Figure 2.14 allows the user to enter values related to the reaction rate and mechanism. First, the user selects whether they want the rate law Equation (2.14) or rate law Equation (2.15) and then the user enters the necessary parameters associated with that rate law. If Equation (2.15) is selected for the rate law, then an added space to enter T_k appears, which isn't shown in Figure 2.14 because it is not necessary for the selected rate law. The reaction coefficients to determine the reaction mechanism are also entered here. An explanation for the reaction coefficients was demonstrated with the example Equation (2.3). If coefficient e was not set to zero, then there would be a space for coefficient f. The mask dynamically adjusts to provide the user with the relevant options. Next, the user selects whether the reaction follows a non-elementary rate law (Equation (2.13)) and enters the required exponential powers (α, β, γ) or selects the elementary rate law form (Equation (2.12)). Finally, the user has the option of entering scaling factors for rate laws, if they do not want to use Equation (2.25). If a box is left unchecked, then a value of 1 is assumed and with a checked box the user can enter their desired value as shown for r_b and r_d in Figure 2.14.

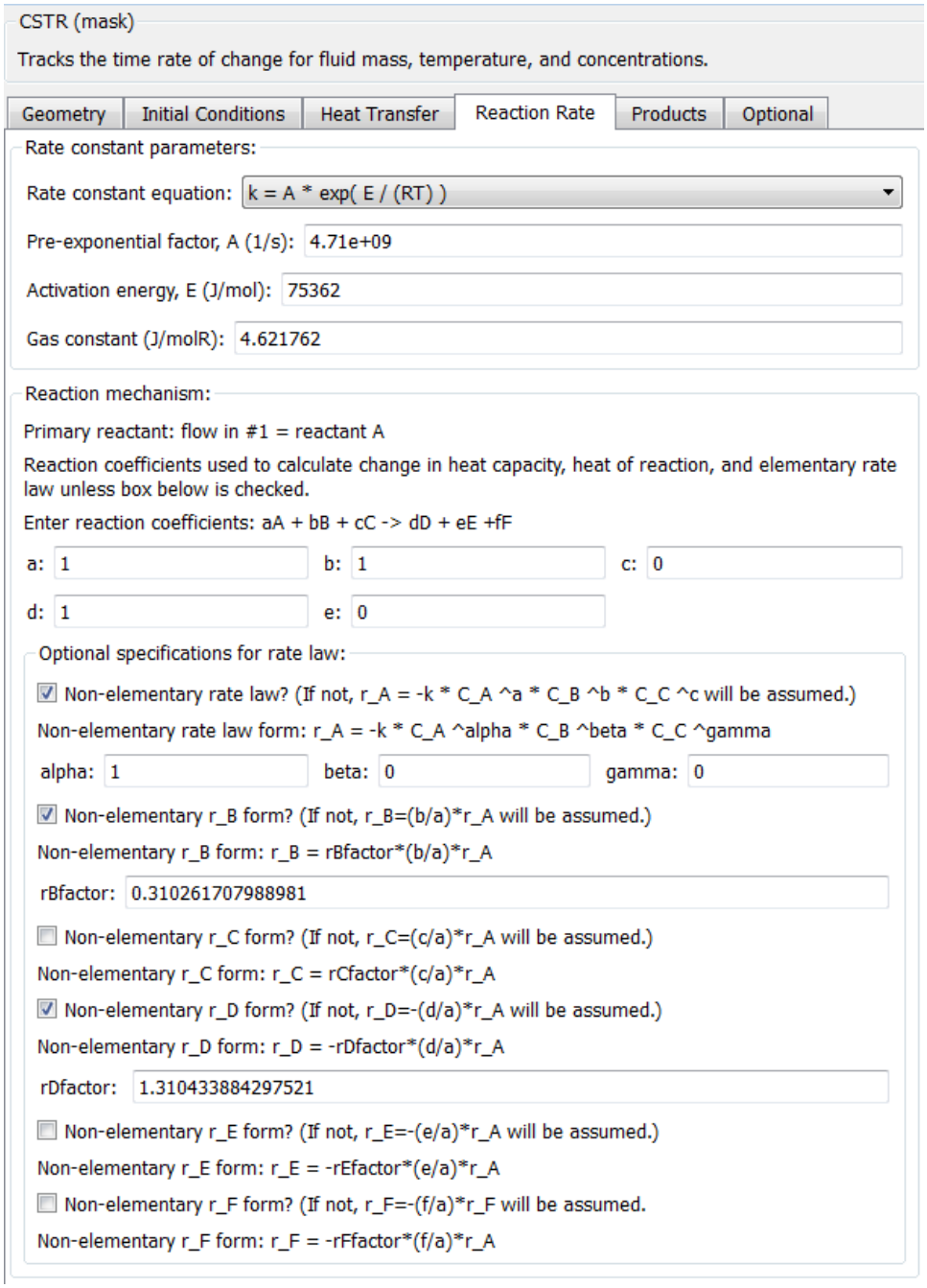

Figure 2.14 CSTR reaction rate parameter tab.

Figure 2.15 allows the user to choose which products are formed. This example allows the user to select only one product because in Figure 2.14, the user indicated only one product. However, if more products were indicated, then more options would appear in the Product E and Product F boxes. The user could also select the fluid type as “Other (user defined)” and then they

would need to enter the heat of formation and specific heat capacity. When an already defined fluid is selected, the fluid properties automatically fill in, as shown for propylene glycol and indicated by the grey background. It is important to note that Product D is the concentration that is tracked in downstream components. In the future, the ability to track more than one concentration could be added, but in the current form only the Product D concentration is tracked after the CSTR.

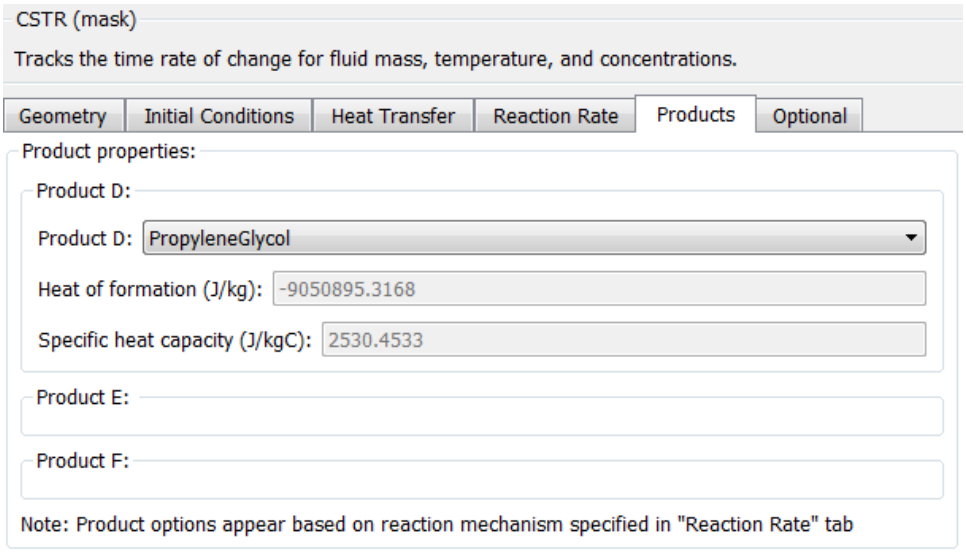

Figure 2.15 CSTR products parameters tab.

Figure 2.16 provides optional values that the user can choose to provide if they check the boxes. If the first box is unchecked, the density will be calculated using Equation (2.18). If the second box is unchecked, the heat capacity of the solution will be calculated using Equation (2.6). If the third box is unchecked, the heat of reaction will be calculated using Equation (2.10). If the fourth box is unchecked, the change in heat capacity of the solution will be calculated using Equation (2.11). The final two boxes are important if a simplified heat of reaction at temperature T equation is desired. Additional details about this simplification are provided in Chapter 4 which explains how sometimes the heat of reaction is assumed to be constant as given in Equation (4.3).

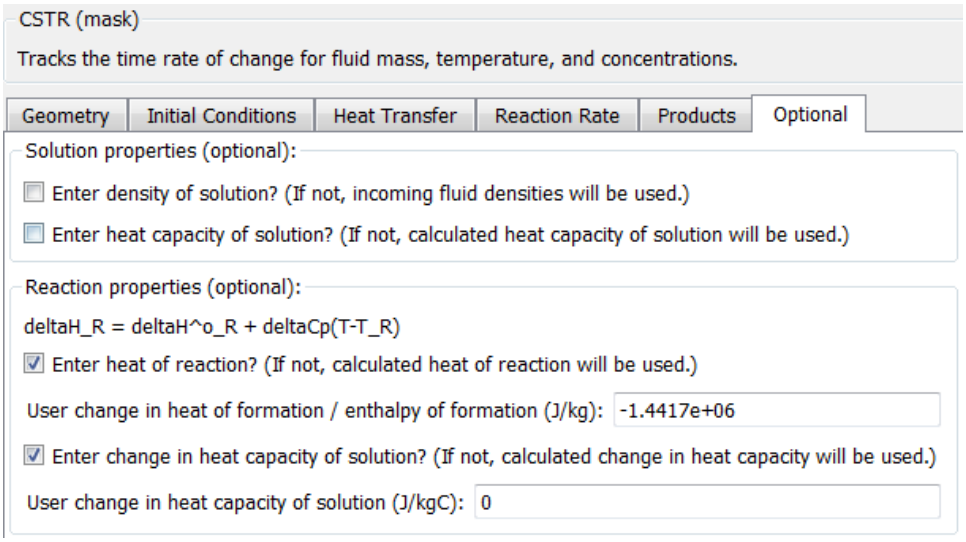

Figure 2.16 CSTR optional parameter tab.

2.3.1.3 Component Inputs and Outputs

The CSTR block has three fluid inlet flows and one outlet flow as shown in Figure 2.17. Each flow contains the temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation of the fluid flow. The additional input is a coolant flow, which must come directly from the CSTR jacket coolant flow outlet. The coolant flow contains the temperature, mass flow rate, and heat capacity of the coolant fluid.

To maintain causality, the mass flow rate out of the CSTR is determined by the downstream block. The CSTR calculates the inlet pressure and sends this pressure to the three upstream inlet flows.

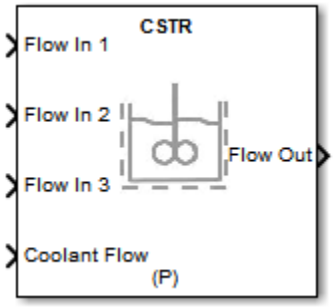

Figure 2.17 CSTR library block showing inputs and outputs.

2.3.1.4 Assumptions

The following assumptions are made for the CSTR.

1. Constant volume – In this model, the volume is dynamic, but restricted to never be less than zero or greater than the total reactor volume. However, the equations used assume that the liquid volume in the tank is constant [18], [36].
2. Perfect mixing – All contents in the reactor are assumed to be well mixed, which means the same concentrations and temperature throughout the reactor and at the reactor outlet [36].
3. Geometry – The tank is assumed to be a cylinder with inlet flows that enter at the top of the tank and outlet flows that leave at the bottom of the tank.
4. Constant pressure – With the liquid volume assumed to be constant, the air volume is assumed to be constant and therefore no measurable pressure change is expected at the inlet. Note that the outlet pressure is calculated by an algebraic pressure equation.
5. No shaft work – The stirring shaft work is neglected because it is assumed to be negligible [18].

2.3.2 Continuous Stirred Tank Reactor Jacket

The CSTR jacket block must always be paired with the CSTR block. The CSTR jacket block provides information about the coolant flow into the CSTR jacket. The CSTR jacket was created as a separate block to make the controlled jacket coolant rate simple to adjust. It is important to note that even though the flows are labeled as coolant, the fluid in the jacket could be warmer rather than colder and serve as a heater for the main reactor.

2.3.2.1 Mathematical Model

The heat transfer from the jacket to the reactor is given by

$$\dot{Q} = \dot{m}_c c_{pc}\{(T_{a1} - T)[1 - \exp(\frac{-UA_r}{\dot{m}_c c_{pc}})]\}, \tag{2.27}$$

where $\dot{m}_c$ is the coolant mass flow rate, c_{pc} is the coolant heat capacity, T_{a1} is the coolant jacket inlet temperature, T is the temperature in the reactor, U is the jacket heat transfer coefficient,

and A_r is the heat transfer area between the jacket and reactor [37]. This is often simplified as given previously in Equation (2.4) for large coolant mass flow rates [37]. Note that the heat transfer calculation given by Equation (2.27) is technically computed within the CSTR block, but logically it makes sense to include with the jacket block.

2.3.2.2 User Inputs

The CSTR jacket block has two user inputs in the GUI. First, the user chooses a fluid type. If the user selects "Other (used defined)", as shown in Figure 2.18, then the user must enter the heat capacity. If the user selects a fluid type in the fluid library, then the heat capacity is automatically generated.

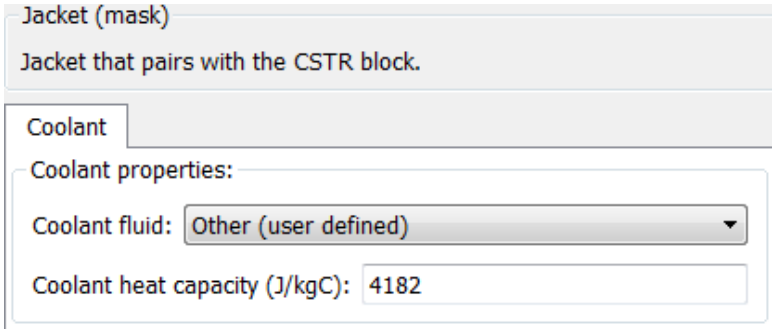

Figure 2.18 CSTR Jacket coolant tab to determine coolant properties.

2.3.2.3 Component Inputs and Outputs

The coolant jacket requires an inlet coolant temperature and coolant mass flow rate. These inlets can be connected to constant blocks or, as will be shown later, a varying control output. The coolant jacket has two outputs. The coolant flow is a bus structure which contains the coolant temperature, coolant mass flow rate, and coolant heat capacity. The coolant flow contains the values that the CSTR block needs to calculate the heat transfer, $\dot{Q}$. The second jacket output is the temperature in the CSTR tank. This value is from the downstream CSTR block. The tank temperature can be left unconnected if desired, but it is included to make control to regulate the tank temperature easier, as will be shown later.

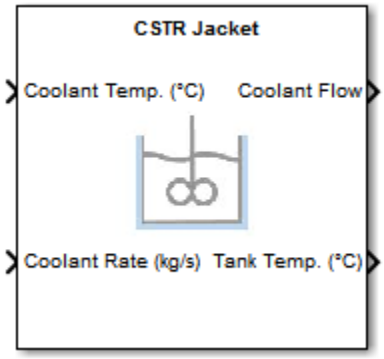

Figure 2.19 CSTR jacket library block showing inputs and outputs.

2.3.2.4 Assumptions

The assumption is made that the heat transfer between the coolant and tank follows the heat transfer equation given in Equation (2.27).

2.3.3 Pump

The pump is used to move fluid in a plant model system.

2.3.3.1 Mathematical Model

The pump calculates the inlet mass flow rate, outlet pressure, and outlet temperature given a known mass flow rate from the downstream block. The mass flow rate of the pump is calculated by

$$\dot{m}_{in} = \rho Q, \tag{2.28}$$

where ρ is the fluid density and Q is the volumetric flow rate given by

$$Q = \sqrt{2g\left[H - 1000\frac{\Delta P}{\rho g}\right]}A, \tag{2.29}$$

$$H = k_{H,1} + k_{H,2} \times \Delta P + k_{H,3} \times PWM, \tag{2.30}$$

$$\Delta P = P_{out} - P_{in}, \tag{2.31}$$

where the coefficients $k_{H,i}$, $i \in \{1,2,3\}$ are found using experimental data, PWM is the duty ratio of the pump, $PWM \in [0,1]$, A is the area at the exit of the pump, and g is gravitational acceleration.

The outlet pressure is calculated with

$$\dot{P} = \frac{\dot{m}_{in} - \dot{m}_{out}}{V\rho\left(\frac{1}{E_{fluid}} + D\frac{1-\nu/2}{tE}\right)}, \tag{2.32}$$

where $\dot{m}_{in}$ is the mass flow rate in, $\dot{m}_{out}$ is the mass flow rate out, V is the volume, ρ is the fluid density, E_{fluid} is the fluid bulk modulus, D is the diameter, ν is the tube Poisson ratio, t is the tube wall thickness, and E is the tube modulus of elasticity.

The outlet temperature is calculated with

$$\dot{T} = \frac{\dot{m}_{in}}{V\rho}\left(T_{in} - T\right), \tag{2.33}$$

where T_{in} is the inlet temperature and T is the temperature of the fluid in the pump. The pump can also calculate the percent efficiency and electrical power consumption given by

$$\eta = k_{\eta,1} + k_{\eta,2} \times WHP + k_{\eta,3} \times PWM + k_{\eta,4} \times WHP^2 + k_{\eta,5} \times PWM^2 + k_{\eta,6} \times WHP \times PWM, \tag{2.34}$$

$$W = \frac{\eta}{100} WHP, \tag{2.35}$$

where the coefficients $k_{\eta,i}$, $i \in \{1,2,...,6\}$ are found using experimental data and $WHP = \rho g Q H$.

2.3.3.2 User Inputs

The pump block has values for the user to enter in a GUI with four tabs. Figure 2.20 provides the option for the user to provide fluid characteristics, if desired. It is expected that the user would usually leave the box unchecked to indicate the fluid is water, or similar enough to water, which would keep the options to enter a new density and bulk modulus hidden. Water is a in the fluid library, which means that the fluid bulk modulus can be looked up. Additionally, there is a two dimensional lookup table for the density of water based on the pressure and

temperature. If water is not used, then a user can either enter a new constant density (checked) or use the incoming fluid density sent in the flow in bus signal (unchecked). If water is not used, then a user can either enter a new fluid bulk modulus (checked) or the bulk modulus of water will be assumed (unchecked). In the future, additional fluid property tables could be added and this fluid parameters tab would be modified to reflect those changes.

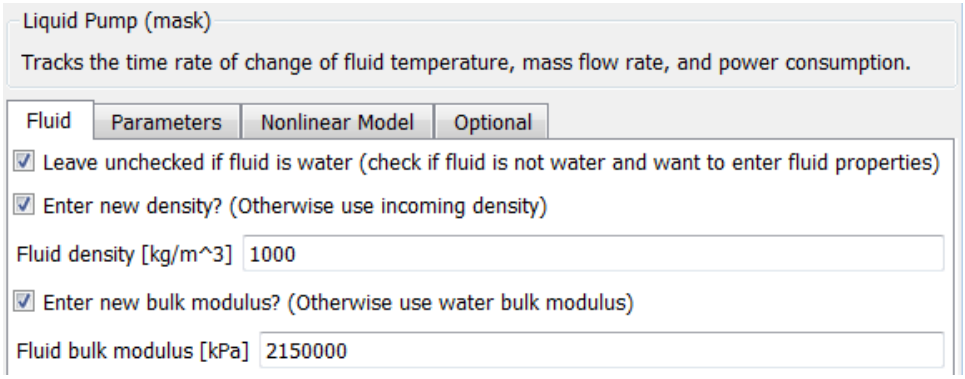

Figure 2.20 Pump fluid parameters tab.

The general parameters tab in Figure 2.21 allows the user to enter the initial outlet pressure and temperature.

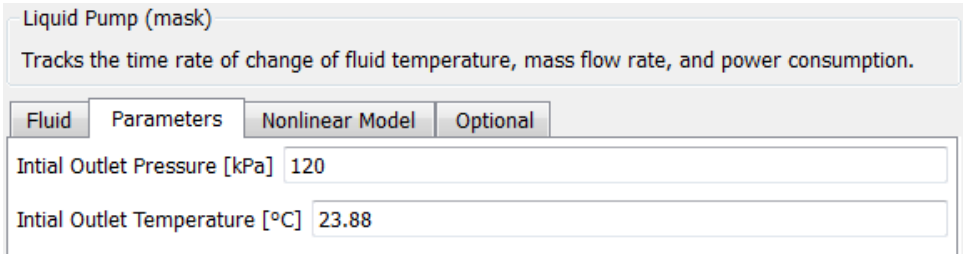

Figure 2.21 Pump general parameters tab.

The nonlinear model parameters tab in Figure 2.22 allows the user to enter the pipe internal diameter, pipe length, tube wall thickness, tube modulus of elasticity, and tube Poisson ratio which are used in the nonlinear pump model.

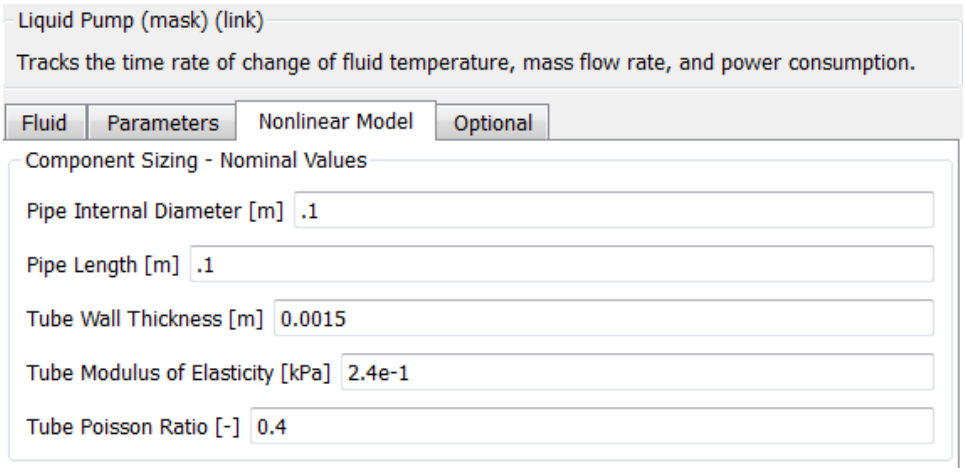

Figure 2.22 Pump nonlinear model parameters tab.

The final optional tab allows the user to artificially scale the mass flow rate by the entered factor as shown in Figure 2.23. The default scaling factor is 1. The option to scale the mass flow rate is provided because the Swiftech MCP35X pump has a lower mass flow rate than may be needed for chemical plant models.

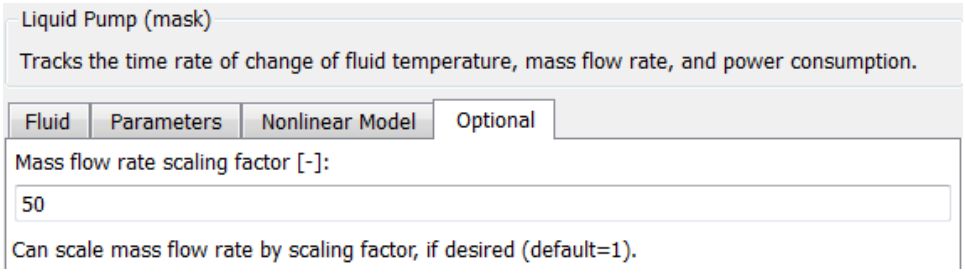

Figure 2.23 Pump optional parameters tab.

2.3.3.3 Component Inputs and Outputs

The pump block has one inlet fluid flow and a duty cycle percent as inputs as shown in Figure 2.24. The inlet fluid flow contains the temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation value. The duty cycle percent should be a numeric value between 0 and 100 representing the operating pulse width modulation for the pump to operate at. The pump block outlet is a fluid flow with the same seven values as in the inlet flow.

To maintain causality, the mass flow rate out of the pump is determined by the downstream block. The pump calculates the inlet mass flow rate and sends this mass flow rate to the upstream block.

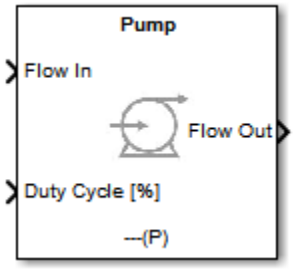

Figure 2.24 Pump library block showing inputs and outputs.

2.3.3.4 Assumptions

This pump is assumed to be a single phase pump. Additionally, this pump is specifically a Swiftech MCP35X pump with a scaling factor of 1 and it is an assumed that the mass flow rate can be scaled by a multiplicative factor [38].

2.3.4 Pipe

There are two pipe versions included in this toolkit. Figure 2.32 shows the standard pipe component (a) which calculates a dynamic outlet pressure and inlet mass flow rate. Additionally, Figure 2.32 shows the secondary pipe component (b) which only calculates a mass flow rate between two pressures. Two pipe versions are included to add flexibility in pipe locations within the model while maintaining the necessary ordering of pressure and mass flow rate calculations.

2.3.4.1 Mathematical Model

The standard pipe version (a) calculates the mass flow rate, outlet pressure and outlet temperature. The mass flow rate is calculated as

$$\dot{m} = \rho u_m A_{cross}, \tag{2.36}$$

$$u_m = \sqrt{\frac{2\left(P_{in} - P_{out} + \rho g \Delta h\right)}{\rho\left(f\frac{L}{D} + K_L\right)}}, \tag{2.37}$$

$$A_{cross} = \frac{\pi D^2}{4}, \tag{2.38}$$

where A_{cross} is the cross sectional area, u_m is the mean fluid velocity, Δh is the change in height, f is the friction factor, L is the length of the pipe, D is the diameter of the pipe, and K_L is the minor loss coefficient [38]. The outlet pressure and outlet temperature are calculated with Equation (2.32) and Equation (2.33), respectively.

The secondary pipe version (b), calculates an algebraic mass flow rate and outlet temperature. The mass flow rate is calculated using Equations (2.36), (2.37), (2.38). However, it is important to note that this secondary pipe does not calculate the outlet pressure, but instead uses the pressure given by the downstream component. The pipe outlet temperature is calculated by Equation (2.33).

2.3.4.2 User Inputs

The standard pipe block (a) has values for the user to enter in a GUI with four tabs. Figure 2.25 provides the option for the user to provide fluid characteristics, if desired, which functions exactly as explained previously for the pump in Section 2.3.3.2.

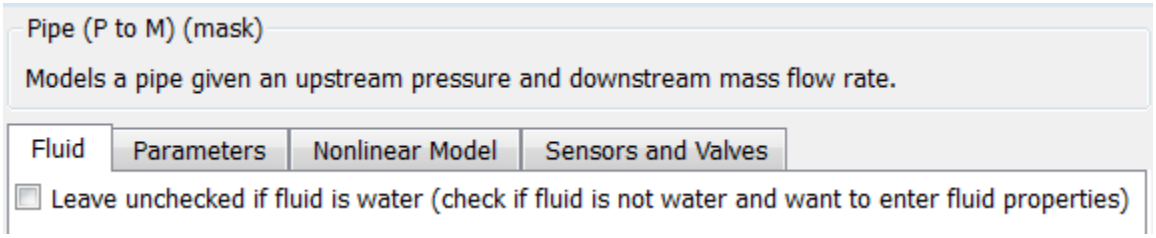

Figure 2.25 Pipe standard version (a) fluid parameters tab.

The general parameters tab in Figure 2.26 allows the user to enter the initial outlet pressure, initial temperature, tube friction factor, and tube minor loss value. The total minor loss value will update automatically based on the user entered tube minor loss value and any added sensors or valves.

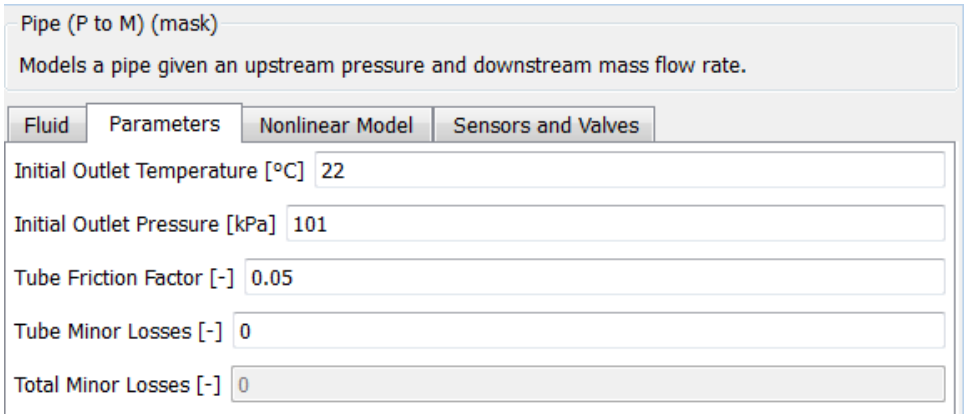

Figure 2.26 Pipe standard version (a) general parameters tab.

The nonlinear model parameters tab in Figure 2.27 allows the user to enter the pipe internal diameter, pipe length, tube wall thickness, tube modulus of elasticity, tube Poisson ratio, and height between the inlet and outlet.

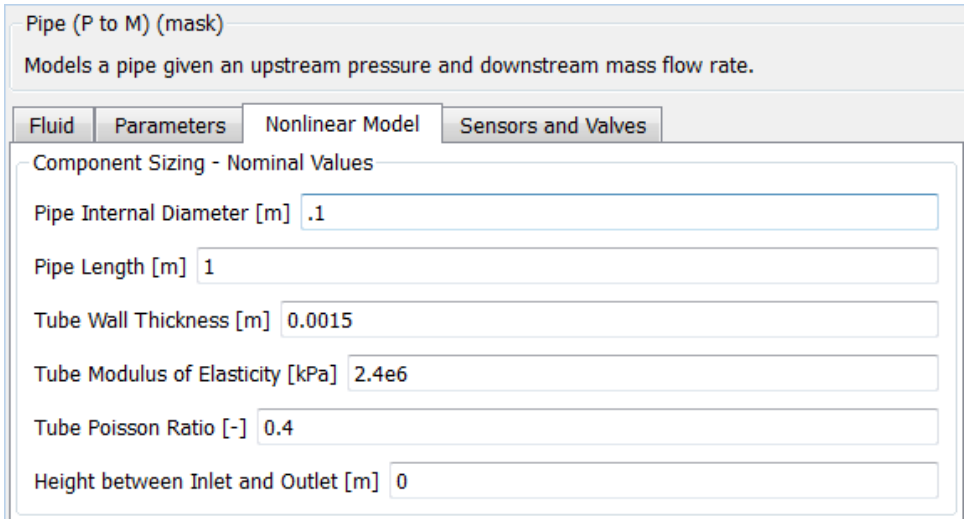

Figure 2.27 Pipe standard version (a) nonlinear model parameters tab.

The sensors and valves parameters tab in Figure 2.28 allows the user to enter the number of mass flow rate sensors, number of pressure sensors, number of temperature sensors, number of ball valves, and number of check valves. The sensors and valves chosen here affect the total minor losses in the pipe shown in Figure 2.26.

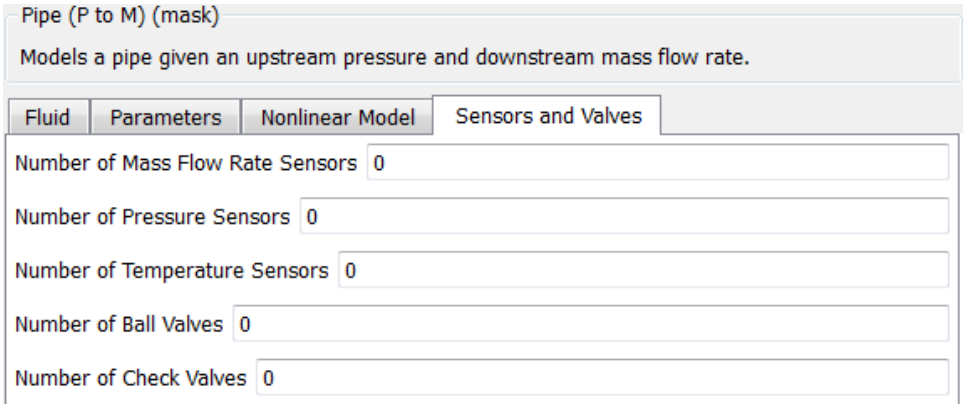

Figure 2.28 Pipe standard version (a) sensors and valves parameters tab.

The secondary pipe block (b) has values for the user to enter in a GUI with three tabs. The general parameters tab in Figure 2.29 allows the user to enter the initial outlet temperature, tube friction factor, and tube minor loss value. The total minor loss value updates automatically based on the user entered tube minor loss value and any added sensors or valves.

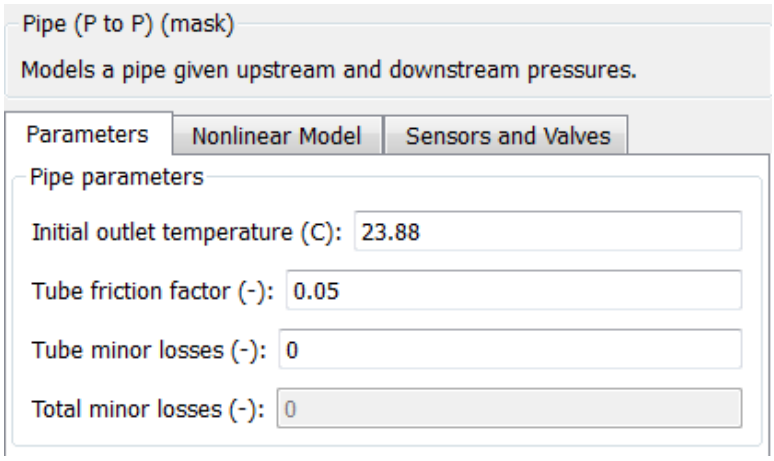

Figure 2.29 Pipe secondary version (b) general parameters tab.

The nonlinear model parameters tab in Figure 2.30 allows the user to enter the pipe internal diameter, pipe length, and height between the inlet and outlet.

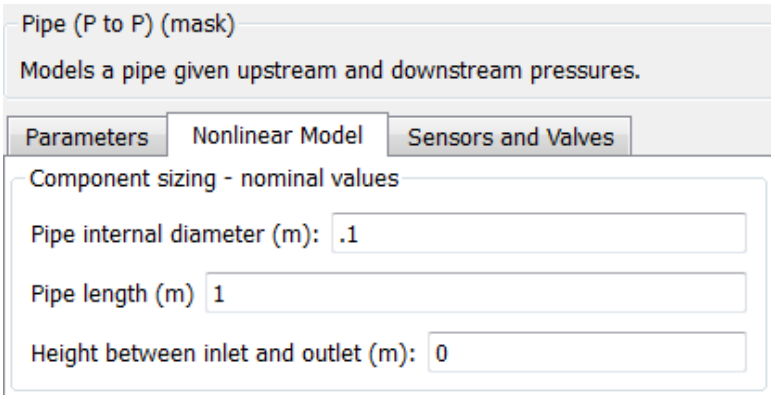

Figure 2.30 Pipe secondary version (b) nonlinear model parameters tab.

The sensors and valves parameters tab in Figure 2.31 allows the user to enter the number of mass flow rate sensors, number of pressure sensors, number of temperature sensors, number of ball valves, and number of check valves. The sensors and valves added selected here affect the total minor losses in the pipe.

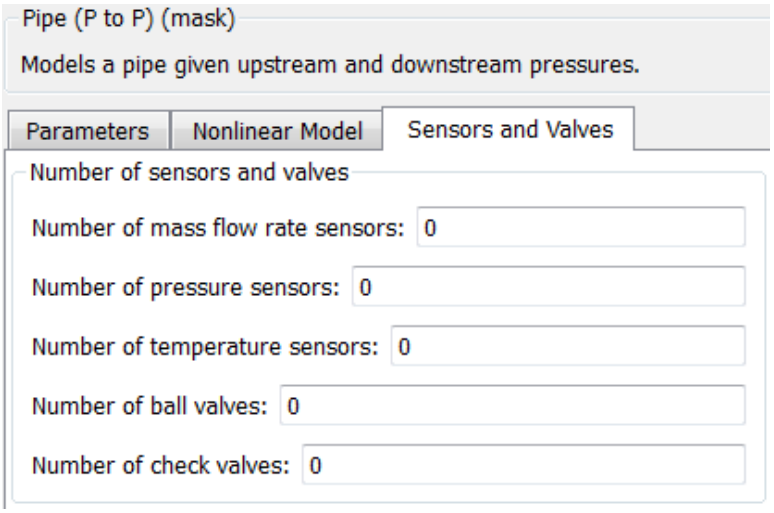

Figure 2.31 Pipe secondary version (b) sensors and valves parameters tab.

2.3.4.3 Component Inputs and Outputs

Both pipe blocks have one inlet fluid flow and one outlet fluid flow as shown in Figure 2.32. The fluid flows contain the temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation value.

To maintain causality, the standard version pipe (a) calculates the inlet mass flow rate and outlet pressure using the mass flow rate out from the downstream block. The standard

version pipe (a) sends the inlet mass flow rate to the upstream block. The secondary pipe version (b) calculates the mass flow rate based on known inlet and outlet pressures. The secondary pipe version (b) sends the mass flow rate to the upstream block.

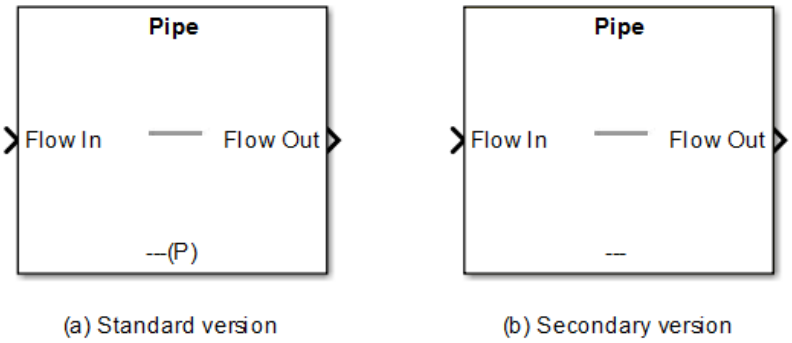

Figure 2.32 The standard pipe version (a) and secondary pipe version (b) showing the inputs and outputs.

2.3.4.4 Assumptions

These pipe models were developed based on first principles and tested with clear PVC 13mm internal diameter and 16mm external diameter piping [38].

2.3.5 Valve

The valve is used to regulate the flow rate through itself.

2.3.5.1 Mathematical Model

The valve calculates an algebraic mass flow rate only. The mass flow rate is given by

$$\dot{m} = \rho Q f_{\%}, \tag{2.39}$$

$$Q = K_v \sqrt{\frac{P_{in} - P_{out}}{SG}}, \tag{2.40}$$

$$SG = \frac{\rho}{\rho_{water}}, \tag{2.41}$$

where $f_{\%}$ is the flow percent, K_v is the flow coefficient, SG is the specific gravity of the fluid, ρ is the fluid density, and ρ_{water} is the density of water [39]. The flow percent, $f_{\%}$, is

determined assuming an equal percentage valve behavior, as shown in Figure 2.33, based off the valve percent open value [40].

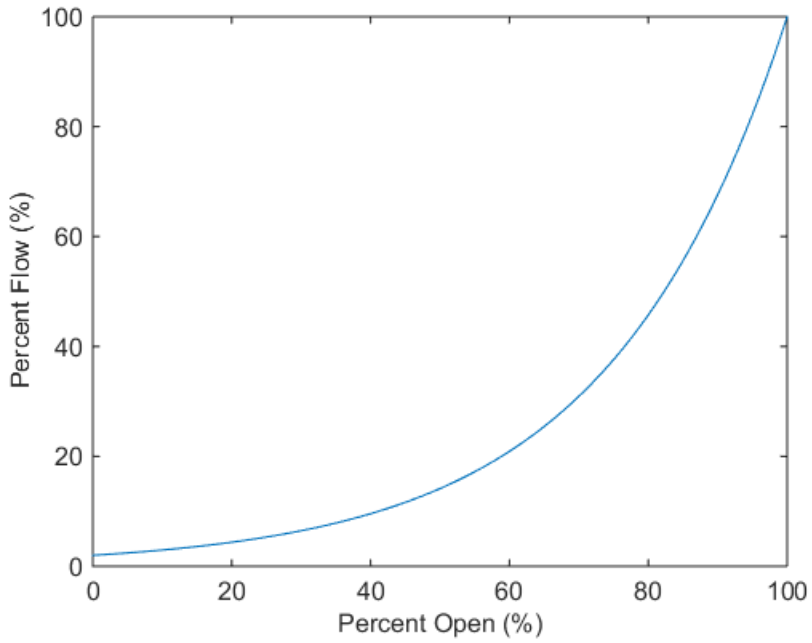

Figure 2.33 Assumed ideal equal percentage valve behavior.

2.3.5.2 User Inputs

The valve block has only one value for the user to enter in a GUI with a single tab. Figure 2.34 shows the space for the user to enter the flow coefficient, K_v. Details for the proper units for K_v and how the volumetric flow rate is calculated, Equation (2.40), are included to help the user as shown in Figure 2.34.

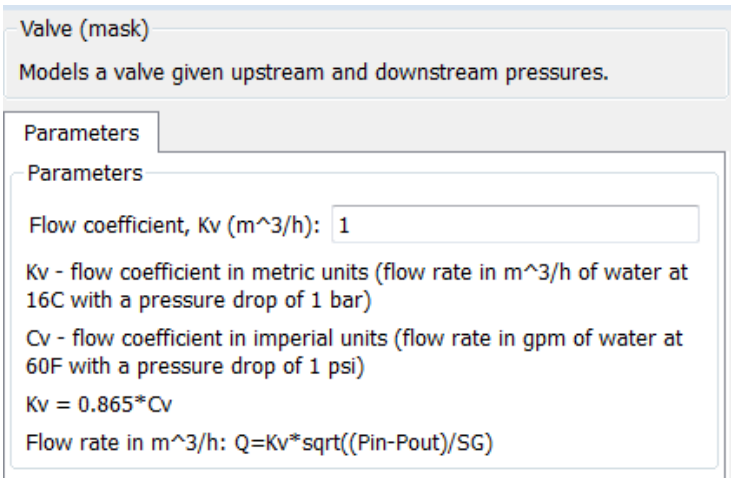

Figure 2.34 Valve parameters tab.

2.3.5.3 Component Inputs and Outputs

The valve block has one inlet fluid flow and one outlet fluid flow as shown in Figure 2.32. The fluid flows contain the temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation value. Additionally, the valve block has a percent open input into which the user can input a constant value or a varying control input. If the user enters a value outside the range of 0-100, the value is saturated to lie within the 0-100 range.

To maintain causality, the valve calculates the mass flow rate based on known inlet and outlet pressures. The valve sends the mass flow rate to the upstream block.

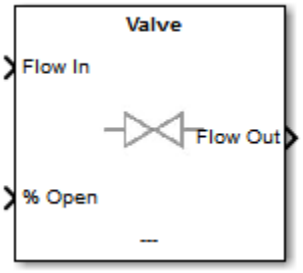

Figure 2.35 Valve library block showing inputs and outputs.

2.3.5.4 Assumptions

The assumption is made that there is no change in temperature because there is a small pressure difference through a valve. Additionally, the valve is assumed to behave as an equal percentage valve. This means that the valve has a linear input output response with a gain of 1.

2.3.6 Mass Flow Rate Source

The mass flow rate source is one option available to start a fluid flow. The other option which is discussed next is a pressure source. The mass flow rate source provides information to the downstream block about the fluid flow including the mass flow rate, but not the pressure.

2.3.6.1 User Inputs

The mass flow rate source block has a GUI with a single tab. Figure 2.36 shows the user interface which requires the user to select a fluid from the library or select "Other (user defined)". If a fluid from the library is selected, such as "Water" as shown in Figure 2.36, then the necessary fluid properties will be automatically filled in. If "Other (user defined)" is selected for the fluid, then the user must provide the density, heat capacity, and heat of formation of the fluid.

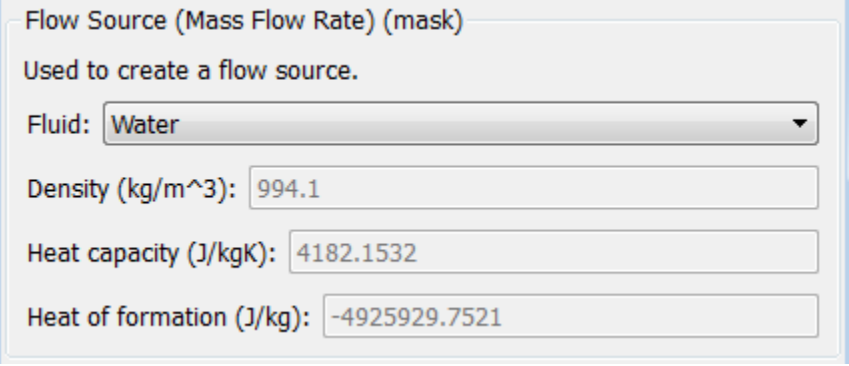

Figure 2.36 Mass flow rate source GUI.

2.3.6.2 Component Inputs and Outputs

The mass flow rate source block has an outlet fluid flow that contains the temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation value. The pressure value is automatically set as -1 and is ignored by the downstream block in order to avoid over defining the model. The mass flow rate source block requires the user to enter numeric values for the temperature and mass flow rate. The inputs can be constant or time varying.

To maintain causality, the mass flow rate source sends a pressure value (-1) that will be ignored by the downstream block and it does not use the value received from the downstream block.

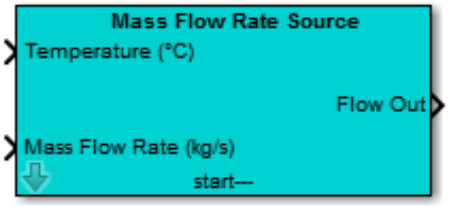

Figure 2.37 Mass flow rate source library block showing inputs and outputs.

2.3.6.3 Assumptions

The assumption is made that the fluid from a mass flow rate source is a pure stream, meaning that the fluid consists of only one chemical substance, such as water, rather than a mixture. Therefore, since concentration and density have the same units of kg/m^3, the concentration is equal to the density provided by the user.

2.3.7 Pressure Source

The pressure source is the second option available to start a fluid flow. The pressure source provides information to the downstream block about the fluid flow including the pressure, but not the mass flow rate.

2.3.7.1 User Inputs

The pressure source block has a GUI with a single tab. Figure 2.38 shows the user interface which requires the user to select a fluid from the library or select “Other (user defined)”. If a fluid from the library is selected, then the necessary fluid properties will be automatically filled in. If “Other (user defined)” is selected for the fluid, then the user must provide the density, heat capacity, and heat of formation of the fluid as shown in Figure 2.38.

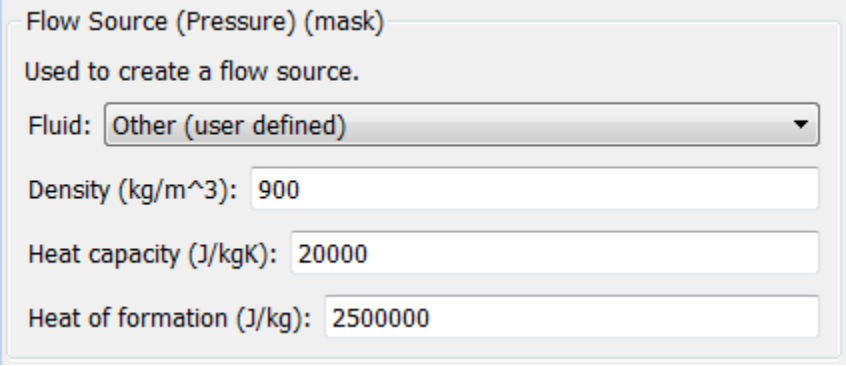

Figure 2.38 Pressure source GUI.

2.3.7.2 Component Inputs and Outputs

The pressure source block has an outlet fluid flow that contains the temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation value. The mass flow rate value is automatically set as -1 and is ignored by the downstream block in order to avoid over defining the model. The pressure source block requires the user to enter numeric values for the temperature and source pressure. The inputs can be constant or time varying.

To maintain causality, the pressure source sends a mass flow rate value (-1) that will be ignored by the downstream block and it does not use the value received from the downstream block.

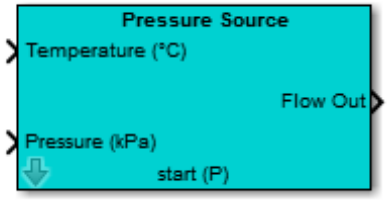

Figure 2.39 Pressure source block showing inputs and outputs.

2.3.7.3 Assumptions

The assumption is made that the fluid from the pressure source is a pure stream, meaning that the fluid consists of only one chemical substance, such as water, rather than a mixture. Therefore, since concentration and density have the same units of kg/m^3, the concentration is equal to the density provided by the user.

2.3.8 Mass Flow Rate Sink

The mass flow rate sink is one option available to end a fluid flow. The other option which is discussed next is a pressure sink. The mass flow rate sink sends the mass flow rate value to the upstream block, but not a pressure value.

2.3.8.1 Component Inputs and Outputs

The mass flow rate sink has an inlet fluid flow that contains the temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation value. The mass flow rate sink block requires the user to enter a numeric value, constant or time varying, for the mass flow rate.

To maintain causality, the mass flow rate sink sends the specified mass flow rate value to the upstream block and ignores all the inlet fluid flow information.

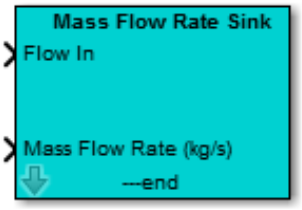

Figure 2.40 Mass flow rate sink block showing inputs.

2.3.9 Pressure Sink

The pressure sink is the other option available to end a fluid flow. The pressure sink sends the pressure value to the upstream block, but not a mass flow rate.

2.3.9.1 Component Inputs and Outputs

The pressure sink has an inlet fluid flow that contains the temperature, mass flow rate, pressure, concentration, density, heat capacity, and heat of formation value. The pressure sink block requires the user to enter a numeric value, constant or time varying, for the pressure.

To maintain causality, the pressure sink sends the specified pressure value to the upstream block and ignores all the inlet fluid flow information.

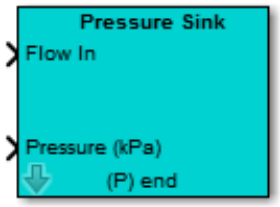

Figure 2.41 Pressure sink block showing inputs.

2.3.10 Support Functions

In addition to the ten main library components previously discussed, the modeling toolkit provides and utilizes four support blocks as shown in Figure 2.42. The data manager is used to monitor and plot desired values. The source, sink, and data sink blocks construct the hidden foundation for all ten main library components. An advanced user could use the source, sink, and data sink blocks to create additional components.

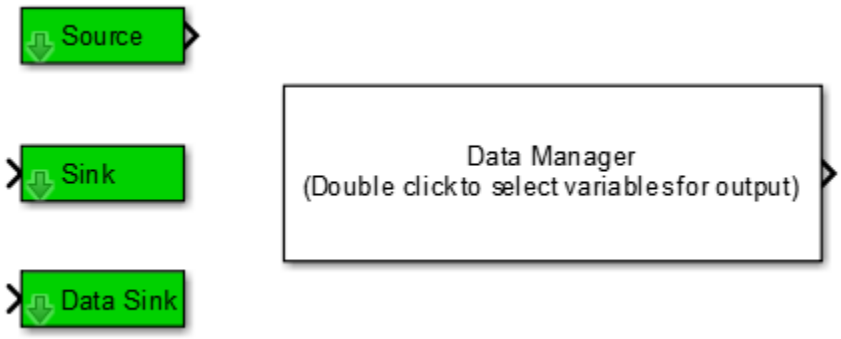

Figure 2.42 Modeling library support functions.

2.3.10.1 Data Manager

The data manager allows the user to access states and other values that are used internally for any blocks found in the model. To use the data manager, the user double clicks on the data manager block and a GUI opens with all the available data signals and the user selects the desired signals as shown in Figure 2.43. The signals can be outputted as a bus structure, as demonstrated in this example, or as a mux vector. Figure 2.44 shows how the selected signals can be easily accessed. The data manager block is what allows intermediate states to be measured since the main blocks do not directly output many values. The only value that is

directly outputted is the tank temperature by the CSTR jacket, but the user may want to monitor other values.

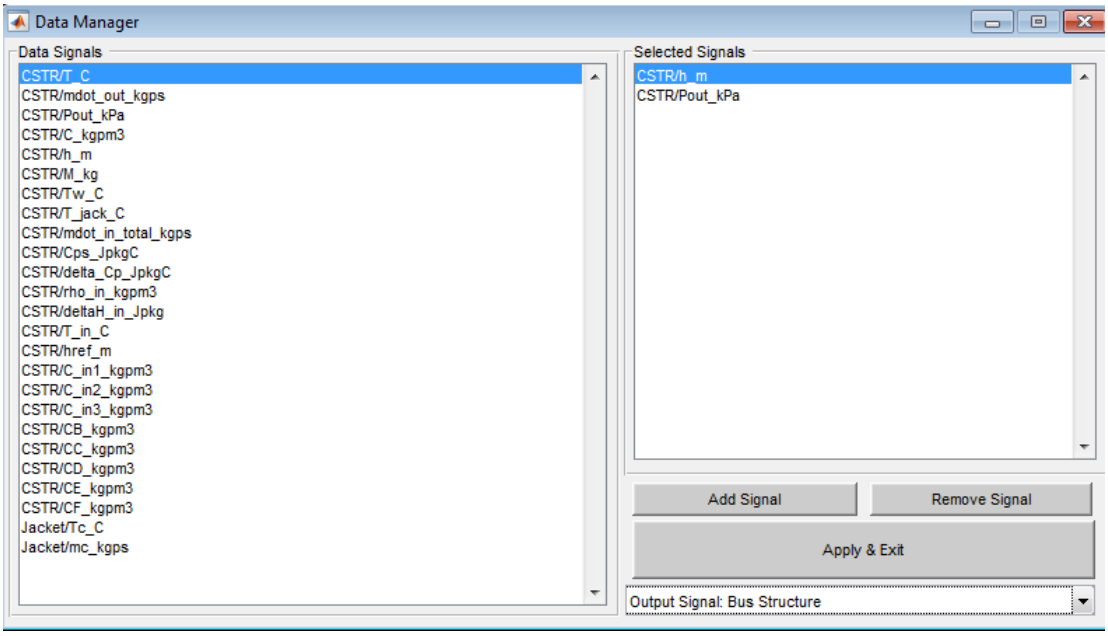

Figure 2.43 Data manager example GUI showing how the desired data signals are selected.

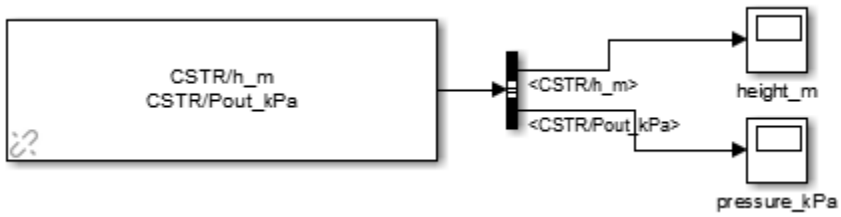

Figure 2.44 Data manager example to demonstrate how information about the CSTR liquid height and outlet pressure can be attained.

2.3.10.2 Data Sink

The data sink block works behind the scenes in each main component block to gather all the information for the data manager. This means that all the information in the data manager was sent into a data sink block. Figure 2.45 provides an example of how the standard pipe sends the outlet temperature, inlet mass flow rate, and outlet pressure to the data sink. The data sink makes it so that those three values are available in the data manager.

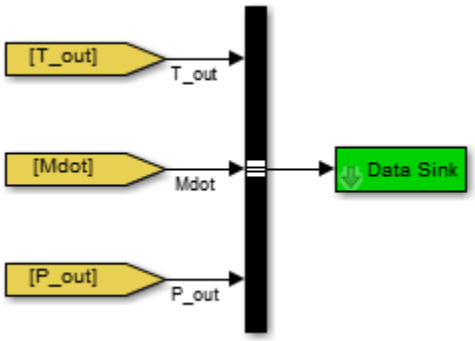

Figure 2.45 Data sink example for the standard pipe showing how the outlet temperature, inlet mass flow rate, and outlet pressure for the pipe will be available in the data manager.

2.3.10.3 Sink

The sink block also works behinds the scenes in each main component block. The sink accepts a single value and sends the value to an upstream block. The number in the sink block label determines which upstream block to send the value to. For example, as shown in Figure 2.46, there are four upstream blocks and so there are four corresponding sink blocks to send a value to each upstream block.

2.3.10.4 Source

The source block also works behinds the scenes in each main component block. The source contains a single value from a downstream block. The number in the source block label determines which downstream block to receive the value from. For example, as shown in Figure 2.46, there is only one downstream block and so the source block is numbered 1. It is important to note that the source and sink names must be enumerated even if there is only one downstream or upstream block.

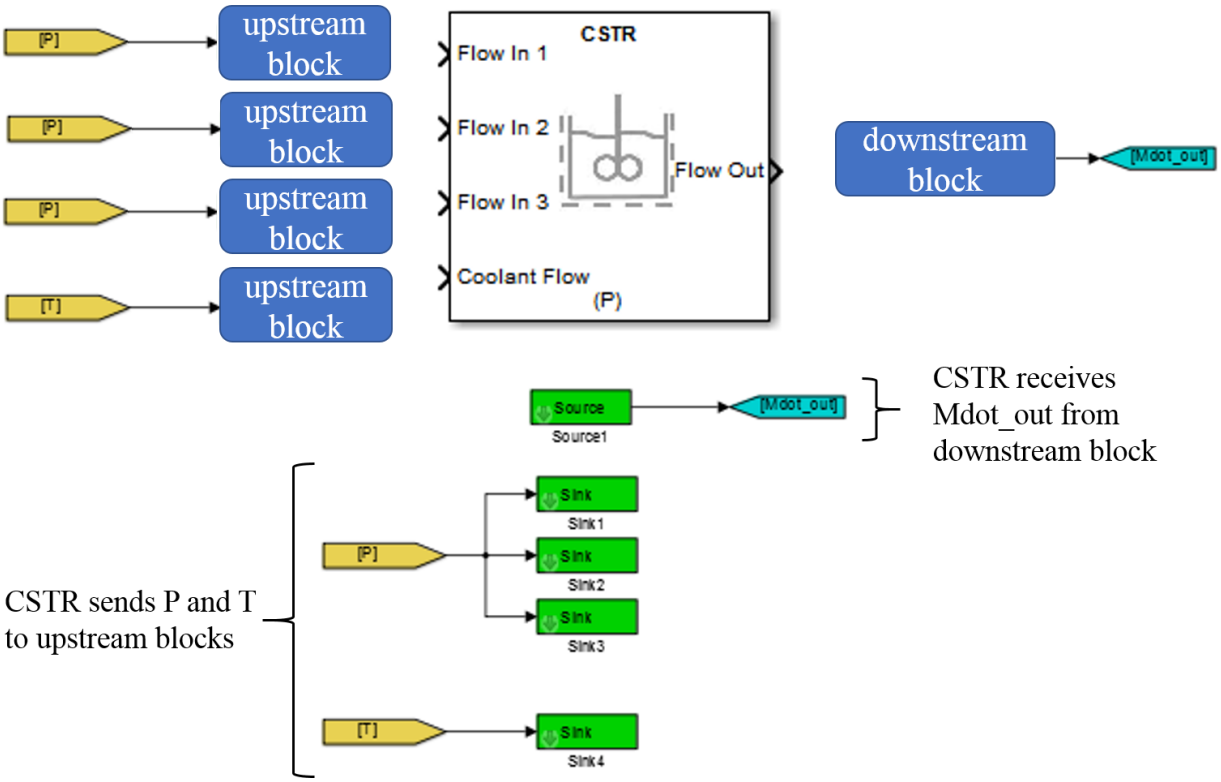

Figure 2.46 Example showing how the sink and source blocks work within the CSTR block.

2.3.11 Assumptions

An important assumption made is that no chemical reactions happen outside of the CSTR. In other words, it is assumed that there are no concentration changes in the pipes, pumps, or valves [36]. Additionally, with how the components are constructed now, it is assumed that the user only needs to track the concentration of the first product downstream because the pipes, pumps, and valves only track a single concentration.

2.3.12 Acknowledgement

The support functions and majority of the pump and pipe model development were done by Justin Koeln, Matthew Williams, and Herschel Pangborn [38].

2.4 Modeling Library Details

2.4.1 Installing Modeling Library

The first time the model library is used, the entire toolkit should be downloaded into the desired installation directory. Then, the automatic installation Matlab script (ChemProcessLib_auto_install.m) should be run. The script will install all the necessary directories for operation of the modeling toolkit.

2.4.2 Adding New Components

New components can be added to the Simulink library. The simplest method to add a new block would be to copy a block that is already in the library and make the necessary changes to create the new desired behavior. However, if the user wants to start from scratch an outline for how to develop a new component block is provided.

1. Develop the framework for the code that resides under the mask first.
2. Include the number of inports equal to the number of inputs the final masked block will have.
3. Include the number of outports equal to the number of outputs the final masked block will have.
4. Include the correct number of sink and source blocks to match the number of inports and outports, respectively.
5. Note that each block needs to have at least one fluid flow in and one fluid flow out to fit in the modeling toolkit framework.
6. Send the desired values to the data sink that the data manager can access later.
7. Highlight all the code, then right click and select “Create Subsystem from Selection”.
8. The subsystem will have the number of inputs and outputs as determined by the number of inports and outports.
9. Right click on the subsystem and select “Create Mask” from within the “Mask” menu.
10. Edit how the block appears by adding code to the “Icon & Ports” tab.
11. Add the desired parameters to the mask.

12. Add any desired initialization to the mask, such as loading fluid property tables.
13. Return to under the mask to edit the remainder of the code since the parameters from the mask are now available.
14. Matlab functions and integrators will likely be used to model dynamic behavior. The initial integrator value will likely be received from the mask.

Chapter 3
Model Validation

3.1 Motivation for Model Validation

In Chapter 2, component models were developed based on first principles. Therefore it is important to test whether, with the simplified modeling assumptions, the components each still capture the correct dynamic behavior. Since there are more detailed chemical models, such as Aspen HYSYS, it is important to ensure that these developed component models contain the necessary transient and steady state behavior [21].

Additionally, since the models developed will serve as the emulated chemical plant for the HIL testing, it is important for the emulated plant behavior to be similar enough to the real plant behavior for the HIL testing to be useful. There are different methods for validating models. For this research, experimental validation was conducted for components for which it was feasible. The validation of some components is left for future work. Furthermore, it would be beneficial in future work to validate an entire plant model in addition to individual component validation.

3.2 Experimental Validation

For this thesis, experimental validation of the chemical concentration within a reactor was conducted. As will be discussed next, experimental model validation of the pipes and pumps had already been conducted, which is why the validation efforts focused on the reactor. In a reactor, the chemical concentrations and temperatures are the main states of interest. To directly validate the temperature state dynamics in a reactor experimentally, an exothermic or

endothermic reaction is needed. Due to lab facility and safety constraints, such as not having a fume hood, exothermic or endothermic reactions were not feasible. However, since the temperature and concentration are very closely related, validating only the chemical concentration is still useful.

3.2.1 Crystal Violet Bleaching Reaction

With the safety challenges and concentration validation goal under consideration, the crystal violet bleaching reaction was carefully selected as a suitable experiment. The reaction does not produce heat or gas, which eliminates many safety concerns. Additionally, it has a low cost experimental set-up and requires no fume hood or specialized protective equipment. The experimental cost of less than $750 is itemized in Appendix B. The reaction combines a crystal violet dye solution and a sodium hydroxide solution (NaOH). The hydroxide molecules react with the crystal violet as shown in Figure 3.1 and the chemical reaction is represented by Equation (3.1)

$$CV^{+} + OH^{-} \rightarrow CVOH. \tag{3.1}$$

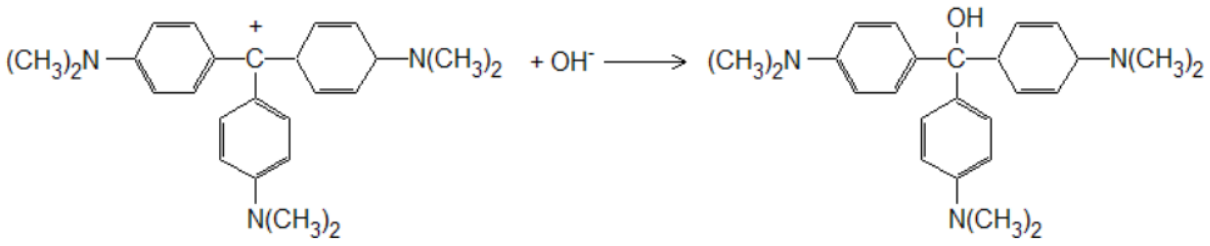

Figure 3.1 Crystal violet and sodium hydroxide chemical reaction structure [41]**.**

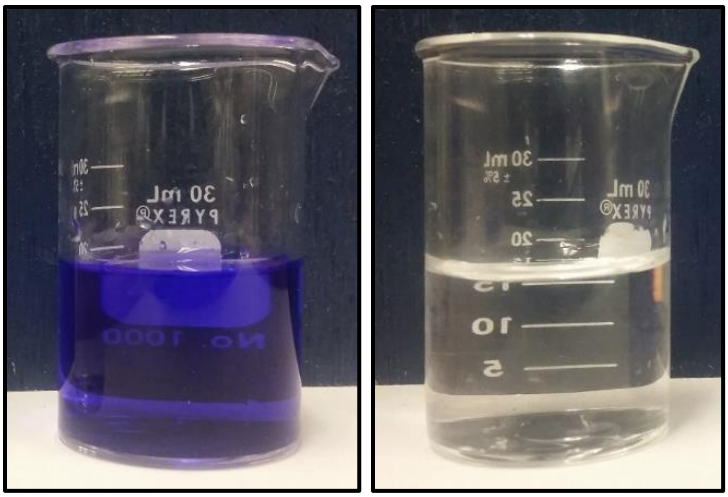

Figure 3.2 Crystal violet reaction as a purple solution at 1 minute (left) and as a colorless solution at 9 minutes (right).

When the reaction begins, the crystal violet dye causes the solution mixture to be a deep purple color. However, as the reaction progresses and the crystal violet ions disappear, the solution gradually transforms from a purple solution to a clear solution as shown in Figure 3.2. The visualization of how the concentration of crystal violet ions decreases is what allows for detection by a photoresistor, rather than a more expensive and time intensive mass spectrometer. The crystal violet reaction follows a first order rate law, which means the unforced dynamic equation which governs the concentration of crystal violet is given by

$$\frac{dC_{CV}}{dt} = -kC_{CV}; C_{CV}(t=0) = C_{CV}(0), \tag{3.2}$$

where C_{CV} is the concentration of crystal violet and k is the reaction rate given by

$$k = k(T) = Ae^{\frac{E}{RT}}C_{OH}. \tag{3.3}$$

This is an Arrhenius equation with E, R, T, C_{OH}, and A. Here A is the pre-exponential factor, E is the activation energy, R is the gas constant, T is the temperature, and C_{OH} is the concentration of hydroxide ions.

3.2.2 Experimental Set-up

The experimental set-up relies on measuring the optical transmissibility as a function of time by shining a light through the beaker with the reagents onto a photoresistor. The primary experimental set-up developed for this thesis to validate how a chemical concentration changes with time is shown in Figure 3.3 and Figure 3.4. The photoresistor is connected to a National Instruments myRIO, an embedded hardware device, which allows for simple data collection using LabVIEW. The photoresistor measures the transmittance with time. To make these experimental results relevant to the chemical modeling, the transmittance needs to be converted to a concentration. First, the percent transmittance is calculated based off the photoresistor voltage for a clear solution. Then, the absorbance is calculated according to

$$Abs = \log(\frac{100}{\%T}), \tag{3.4}$$

where Abs is the absorbance and $\%T$ is percent transmittance. Then, using Beer's Law, the concentration of crystal violet can be calculated using

$$C_{CV} = \frac{Abs}{\varepsilon b}, \tag{3.5}$$

where C_{CV} is the concentration of crystal violet, Abs is the absorbance, ε is the molar extinction coefficient, and b is the path length of the solution [41].

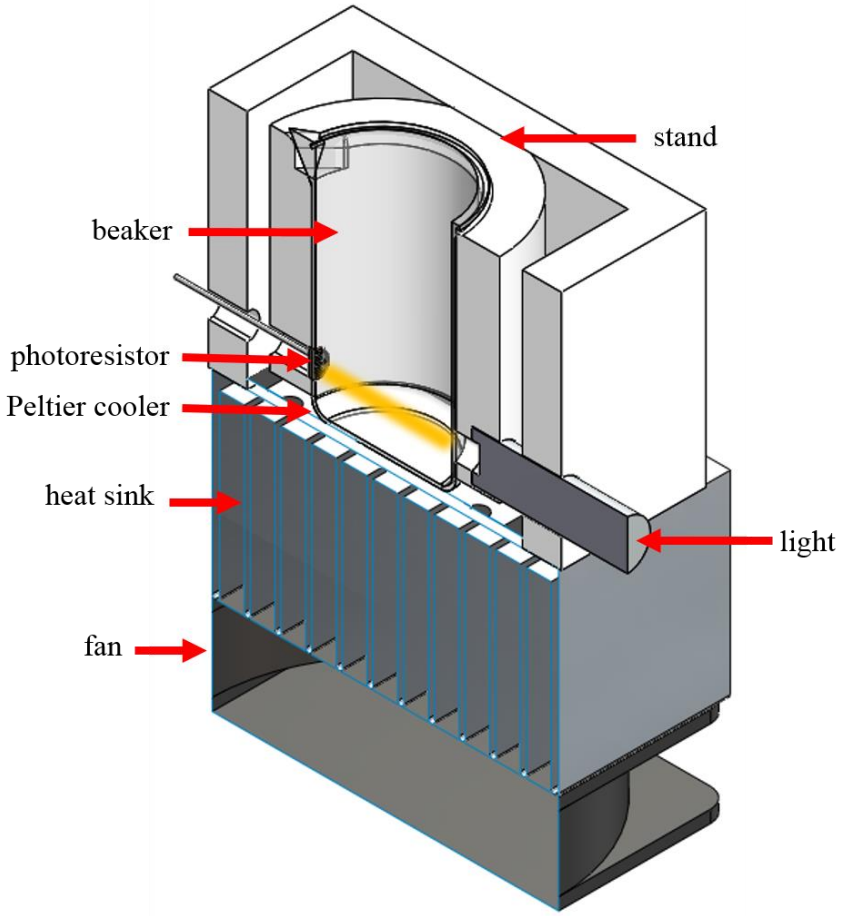

Figure 3.3 SolidWorks model of experimental set-up to measure how the chemical concentration of crystal violet changes with time using a photoresistor.

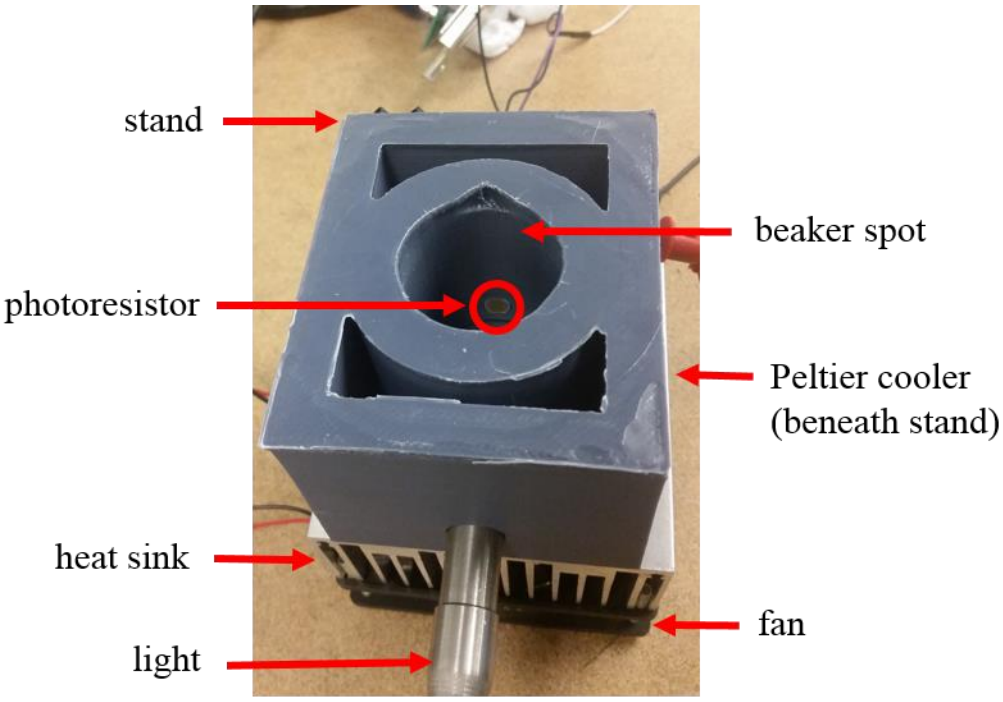

Figure 3.4 Experimental set-up to measure how the chemical concentration of crystal violet changes with time using a photoresistor.

An important aspect of the reaction rate, Equation (3.3), is that the rate depends on temperature. The temperature dependence provides an easily adjustable process parameter to test whether the models are capturing the effect of temperature correctly. To control the temperature, the experiment is conducted on a Peltier cooler assembly like the one shown in Figure 3.5. The Peltier cooler is a thermoelectric device that can heat or cool the reagents.

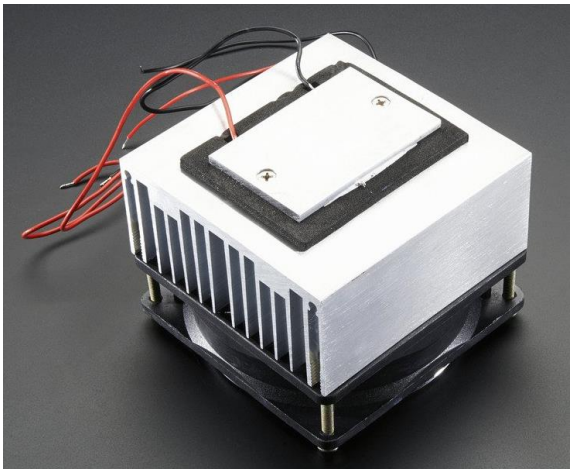

Figure 3.5 Peltier thermo-electric cooler module and heat sink assembly used [42]**.**

3.2.3 Temperature Dependence

Different constant reagent temperatures were tested experimentally and in simulation to determine whether Equation (3.2) adequately captures how the concentration of crystal violet changes with time for various temperatures. The constant temperatures were set by adjusting the applied voltage to the Peltier cooler. According to Equation (3.2) and Equation (3.3), warmer temperatures should result in a smaller time constant and faster decrease in crystal violet concentration which is what Figure 3.6 shows. Figure 3.6 validates that the Arrhenius equation which governs how the reaction rate depends on temperature is capturing the dynamic behavior accurately. This is an important assumption in the CSTR model, which is why it was important to validate how a reactant concentration is affected by temperature experimentally. A complete and detailed procedure for how the experimental results in Figure 3.6 were generated is provided in Appendix B.

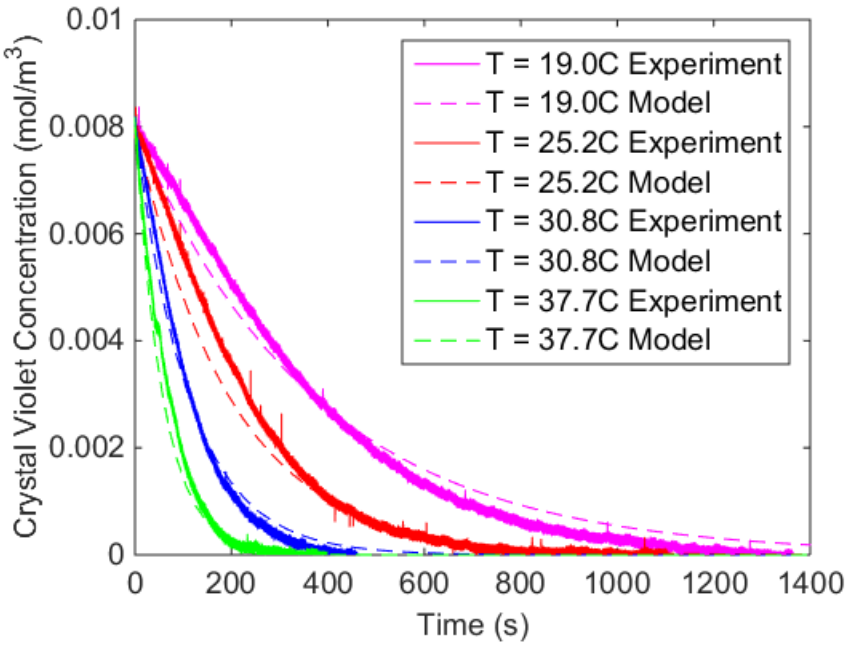

Figure 3.6 Crystal violet concentration versus time for experimental data and model results for different reaction temperatures.

3.2.4 Dynamic Response

The dynamic response of how the crystal violet concentration changes with time if more crystal violet is added throughout a trial at a constant temperature was also tested. Figure 3.7 shows how the crystal violet concentration evolves with additional crystal violet solution added at 400 seconds and 800 seconds. Adding more crystal violet solution is equivalent to resetting the initial condition for the crystal violet concentration in Equation (3.2) or observing the system impulse response. The ability to correctly model how the concentration varies with changing inputs relates directly to how the concentration in a CSTR varies depending on the inlet flow concentrations. This experiment is a simplified version because there is not a continuous flow in and no flow out, but it does increase confidence that the concentration dynamics are captured by the model. Additionally, this test is the main reason why the experimental set-up was created with no lid or cover so that additional reagents could be added throughout the trial. Other off the shelf experimental apparatuses, such as a spectrophotometer, would not have allowed additional reagents to be added during a trial. A detailed procedure for how the experimental results in Figure 3.7 were generated is provided in Appendix B.

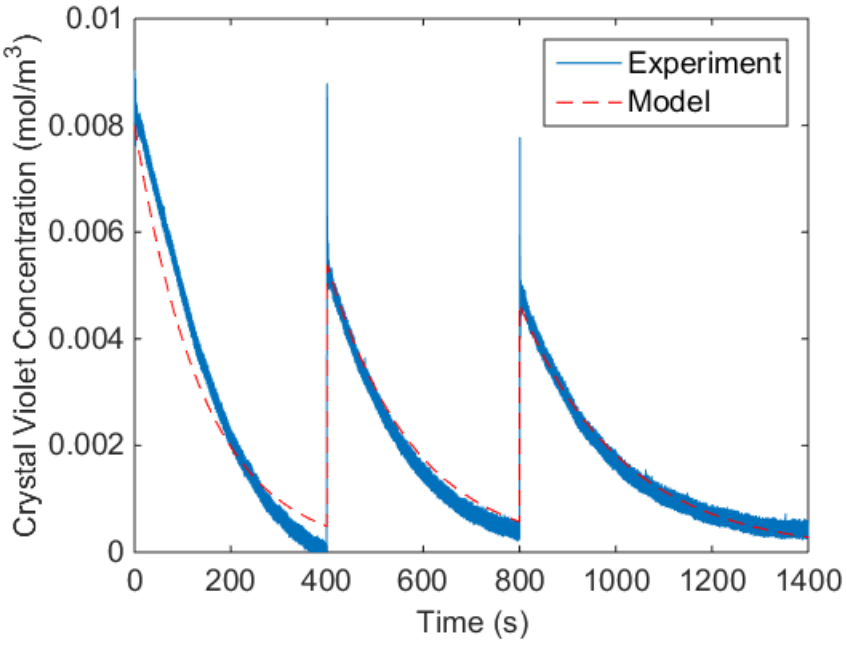

Figure 3.7 Crystal violet concentration versus time for experimental data and model results with added step inputs of additional crystal violet solution at 400 seconds and 800 seconds.

3.3 Previous Model Validation

Model validation of the pipe and pump was conducted experimentally on the testbed shown in Figure 3.8 and described in [38]. The pipe models were developed for flexible PVC tubing of 13 mm internal diameter and 16 mm external diameter. Validation of the pipes was incorporated with the entire fluid thermal model validation.

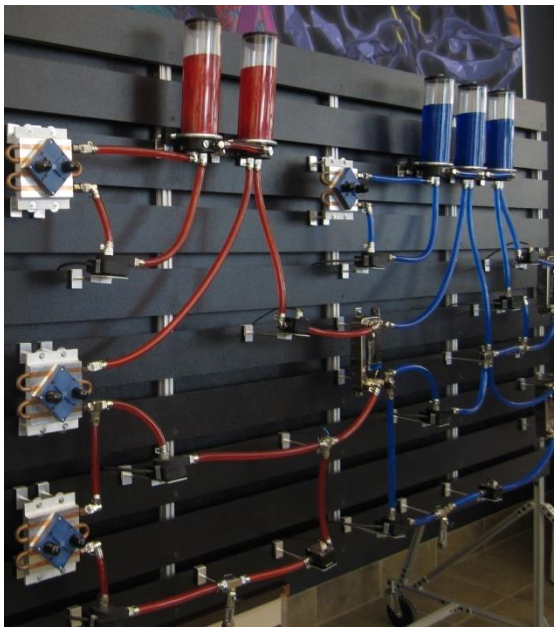

Figure 3.8 Experimental fluid thermal management testbed [38]**.**

For the pump validation, the nominal pump model coefficients were identified from the data of a Swiftech MCP35X pump for pulse width modulation between 20% and 60% duty cycle. The experimentally obtained pump head map is

$$H = k_1 + k_2 \Delta p_p + k_3 \omega, \tag{3.6}$$

where the coefficients $k_i, i \in \{1,2,3\}$ are found using experimental data, $\Delta p_p = p_{out} - p_{in}$ is the pressure differential across the pump, and ω is the pump speed. An example pump head map is provided in Figure 3.9. However, since this pump model has lower flow rates than are needed for many chemical plant systems, the pump model has the added flexibility discussed in Chapter 2 to scale the mass flow rate in approximating the behavior of pumps of different sizing. It is

important to note, however, that this particular pump model has only been experimentally validated for the Swiftech MCP35X pump.

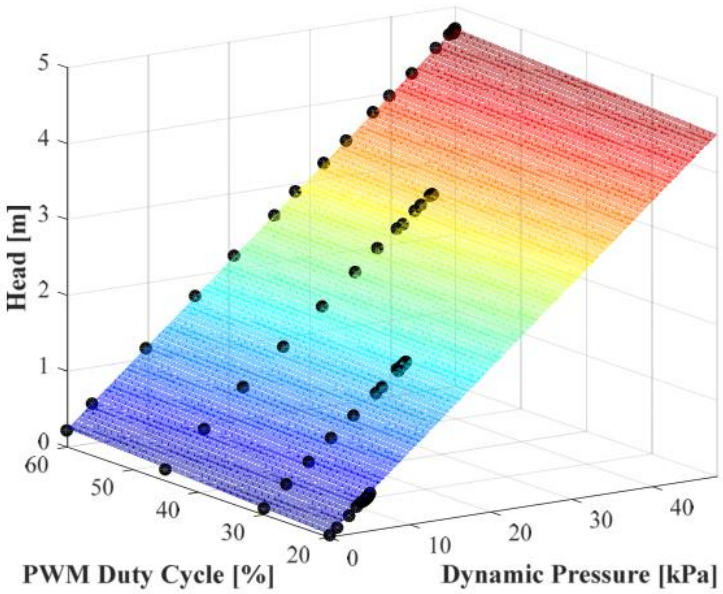

Figure 3.9 Example pump head map [38]**.**

For the overall model validation, separate experimental tests were used to validate the hydrodynamic and thermodynamic domains. The hydrodynamics were validated using rapid steps in pump speed while the slower thermodynamics were validated with slower steps in pump speed and heat load [38]. Figure 3.10 shows the rapid steps in pump speed used for hydrodynamic validation. The experimental results and models, labeled nonlinear graph and linear graph, shown in Figure 3.11 convey how accurate the pump model is for the pump mass flow rate and outlet pressure.

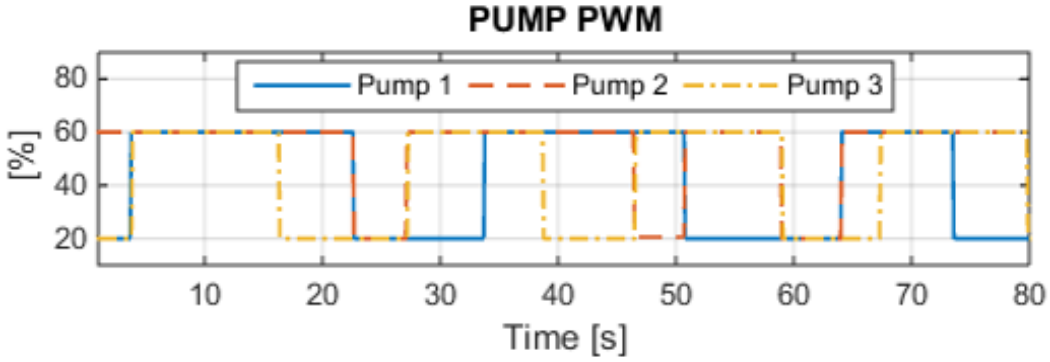

Figure 3.10 Pump PWM inputs for hydrodynamic validation [38]**.**

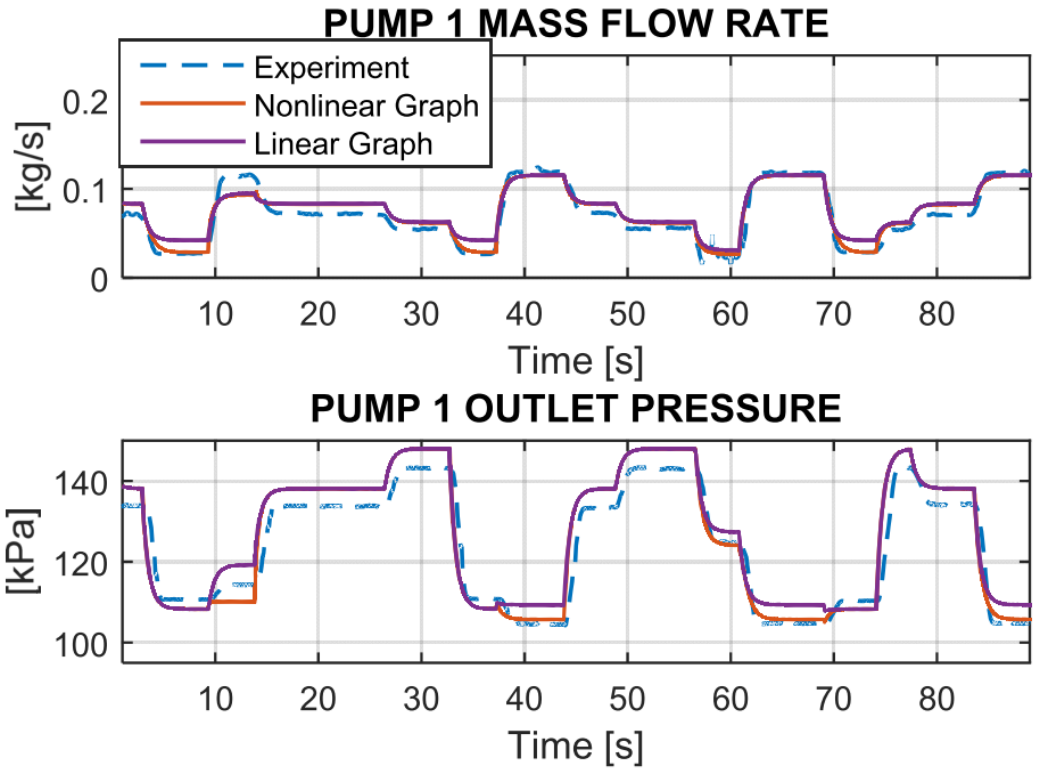

Figure 3.11 Experimental and model values for pump 1 mass flow rate and outlet pressure for hydrodynamic validation [38]**.**

3.4 Future Model Validation

In the future, the valve model, CSTR temperature model for an exothermic or endothermic reaction, CSTR jacket, and a full plant model should be validated. The valve model is assumed to behave as an equal percentage valve with the percent flow related to the percent open as previously shown in Figure 2.33. Additionally, the valve model interacts with the neighboring components in the same way as the pipe, except for the alternative method for calculating the mass flow rate based on the percent open rather than pipe factors such as length and friction factor. Additional resources that would mitigate safety concerns would allow for the validation of the CSTR temperature component together with the CSTR jacket. Finally, a full plant model of a complex chemical reaction would be useful to validate all the component interactions. However, since data for a chemical process in a real plant was unavailable, a complex reaction found in the literature was used instead as shown in Chapter 4.

Chapter 4
Process Control Example

This chapter demonstrates the capabilities of the modeling toolkit developed in Chapter 2. First, it shows how the toolkit can be used to model a chemical plant using the formation of propylene glycol as an example chemical process. Additional capabilities of the modeling toolkit will be explored using the propylene glycol plant model as an example system. These capabilities and features include showing interesting model dynamics, an analysis of model sensitivity, and simple process control implemented in simulation.

4.1 Propylene Glycol Reaction Model

To demonstrate how a chemical process can be modeled using the library toolkit, a model of the formation of propylene glycol in a CSTR will be developed. Propylene glycol is a clear, viscous, colorless liquid that has a slightly bittersweet taste and is nearly odorless [43]. Propylene glycol is unique among glycols because it is safe for humans and is found in alcoholic beverages, frostings, frozen dairy products, seasonings, and nuts, as well as cosmetics, paints, and liquid detergents [43]. Since propylene glycol has so many uses as a known antimicrobial and effective food preservative, it is heavily produced globally and in the United States [43], [44]. In 2014, Dow Chemical produced 920,000 mt/y and Lyondell Chemical produced 410,000 mt/y of propylene glycol, which combined is slightly over 43% of the total global propylene glycol production of 3,063,000 mt/y [44]. The large production of propylene glycol and widespread uses in consumer products, makes it an interesting case study. Additionally, the formation of propylene glycol is a complicated enough model, with three inlet flows, to be representative of general processes.

The propylene glycol reaction model starts with water in the CSTR initially. Then, propylene oxide, water, and methanol all continuously flow into the reactor. The product, propylene glycol, as well as unreacted reactants continuously flow out as shown in Figure 4.1. The reaction takes the form

$$\text{propylene oxide} + \text{water} + \text{methanol} \rightarrow \text{propylene glycol}. \tag{4.1}$$

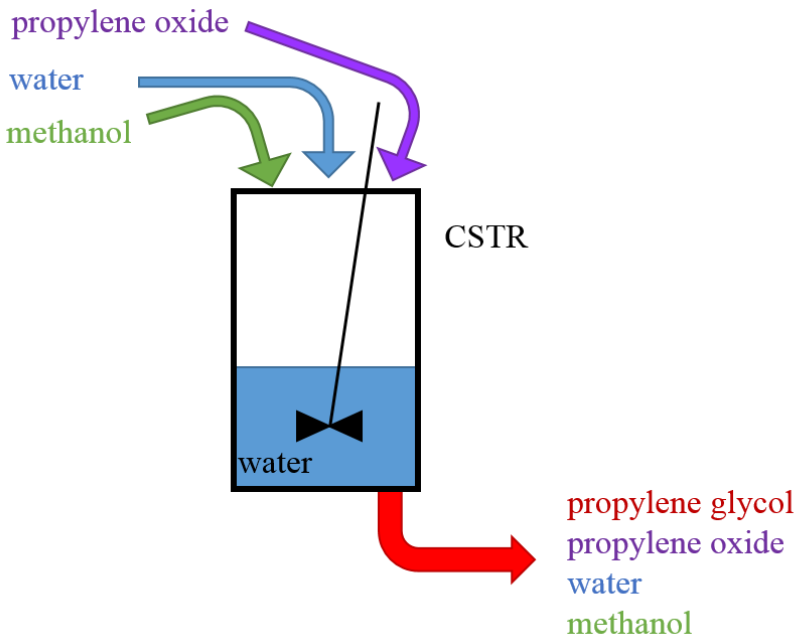

Figure 4.1 Propylene glycol reaction schematic showing the reagents and initial setup with water starting in the CSTR.

In chemical plants, it is common practice to control flow rates using a pump and valve as shown in the P&ID in Figure 4.2 [19], [45]. The convention of using pumps and valves was followed for developing the full plant model shown in Figure 4.3. From the Simulink model, pump duty cycles and valve percentages open are visible. Other parameters that may be of interest to understand the size and important properties about the process can be found in Table 4.3.

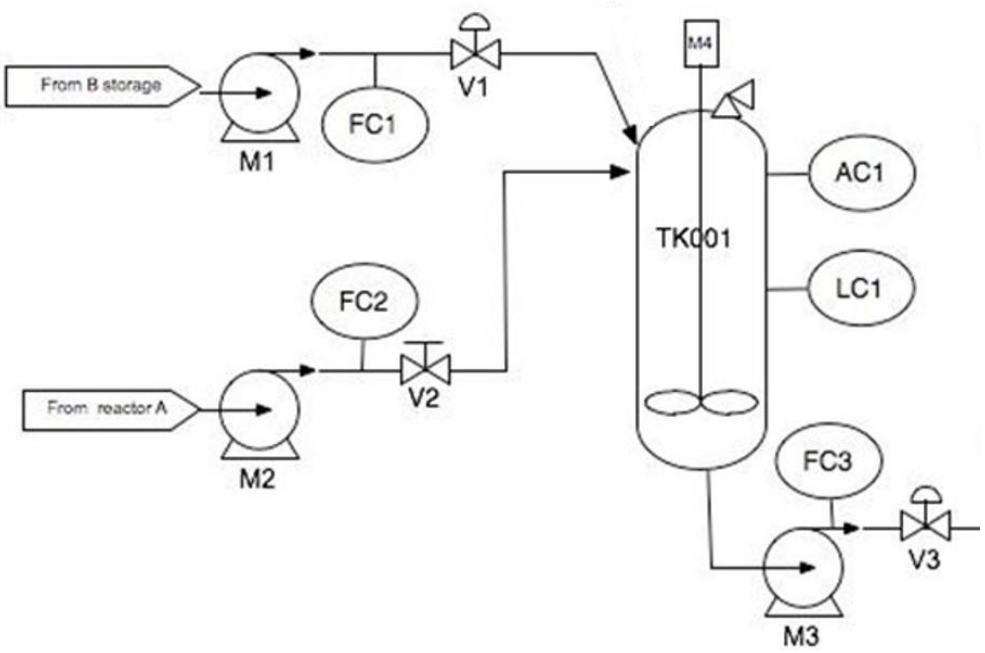

Figure 4.2 Example P&ID for a reaction process with two flows into a CSTR [45]**.**

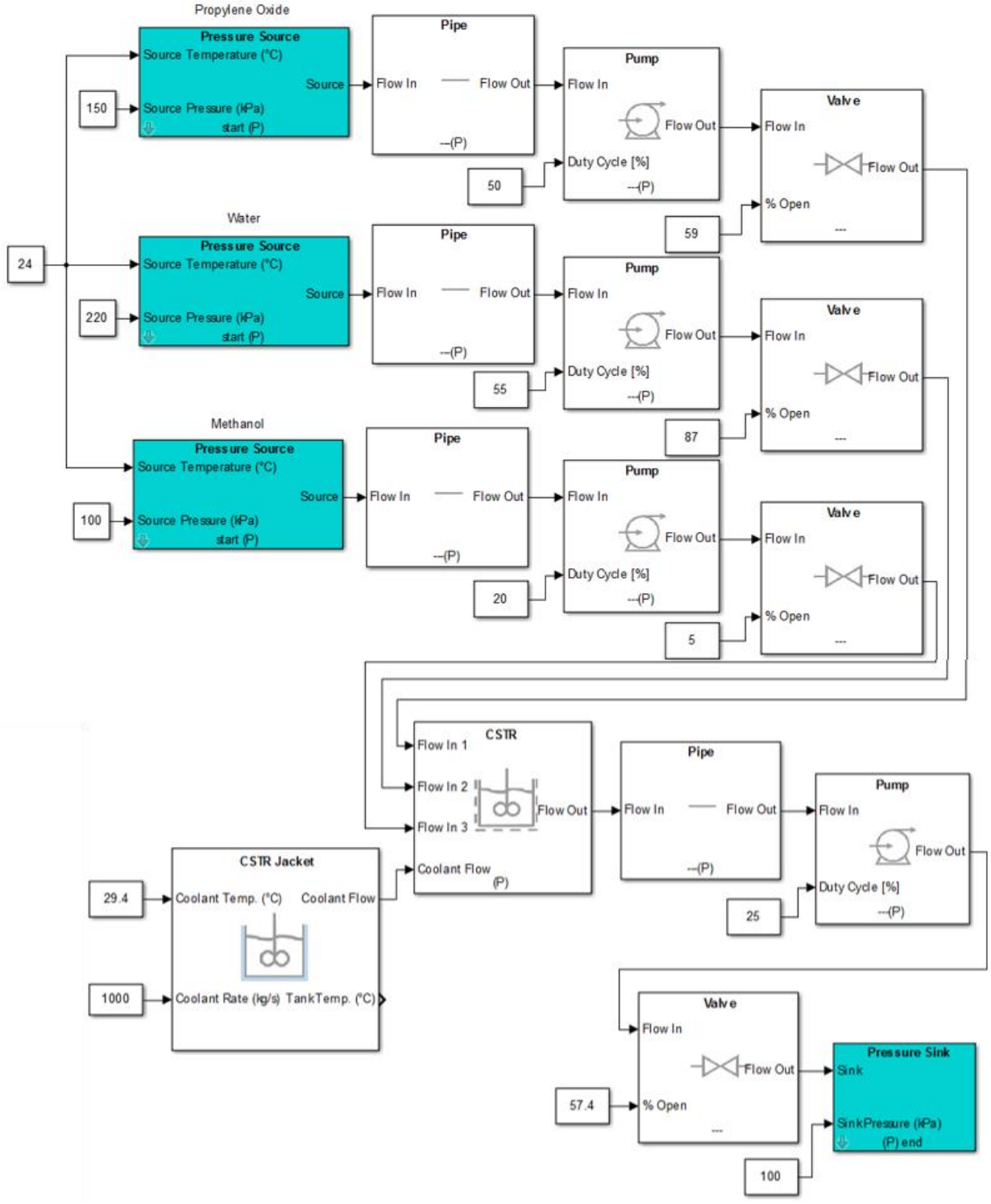

Figure 4.3 Propylene glycol reaction plant model in Simulink.

Table 4.3 Select parameters of interest for propylene glycol reaction Simulink plant model in Figure 4.3.

Parameter	Numeric Value
$\dot{m}_{out}$ propylene oxide valve	2.9 kg/s
$\dot{m}_{out}$ water valve	11.4 kg/s
$\dot{m}_{out}$ methanol valve	0.08 kg/s
$\dot{m}_{out}$ product valve	14.4 kg/s
d all pipes	0.1 m
d CSTR tank	1.68 m
h liquid in CSTR	2.44 m
h CSTR tank	4.00 m
U heat transfer coefficient between CSTR jacket and CSTR	1754 J/sm^{2}C

4.1.1 Propylene Glycol Reaction Model Results

The results from the plant model in Figure 4.3 can be compared with the example problem results from Fogler's textbook, Elements of Chemical Reaction Engineering [18], [46]. The textbook results provide plots only for how the reaction temperature and propylene oxide concentration change with time [46]. The textbook results and plant model results for these states are in good agreement as shown in Figure 4.4 and Figure 4.5. Additional model results include the concentration of the product, propylene glycol, as shown in Figure 4.6. The other reactant concentrations are shown in Figure 4.7 for water concentration and Figure 4.8 for methanol concentration. The textbook results did not include results for these other concentrations for comparison, but the good agreement between the temperature and propylene oxide concentration is reassuring.

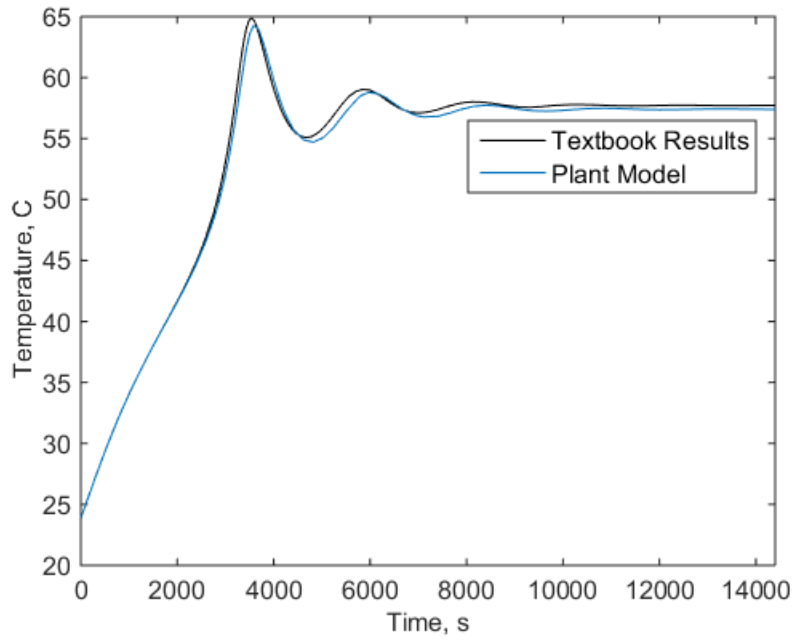

Figure 4.4 Reaction temperature versus time for propylene glycol reaction simulation model results and textbook results [46]**.**

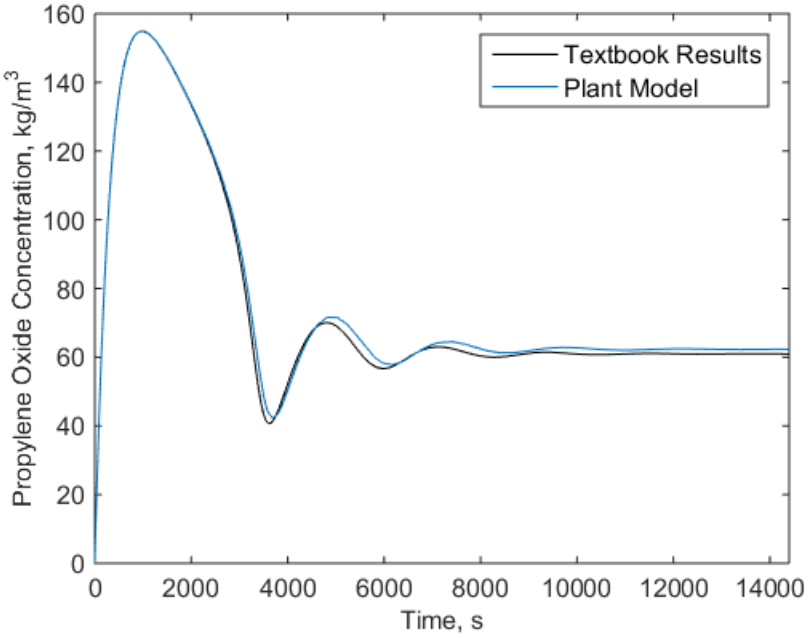

Figure 4.5 Propylene oxide concentration versus time for propylene glycol reaction simulation model results and textbook results [46]**.**

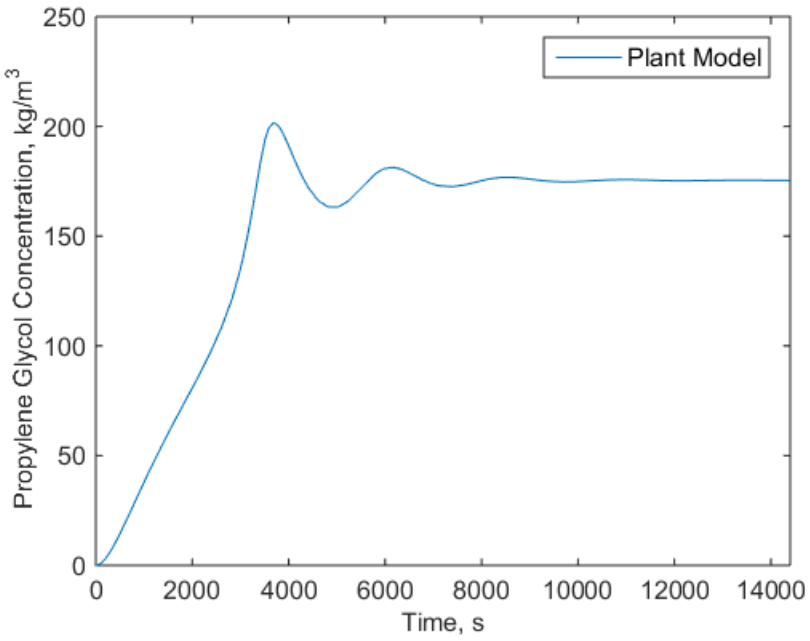

Figure 4.6 Propylene glycol concentration versus time for propylene glycol reaction simulation model results.

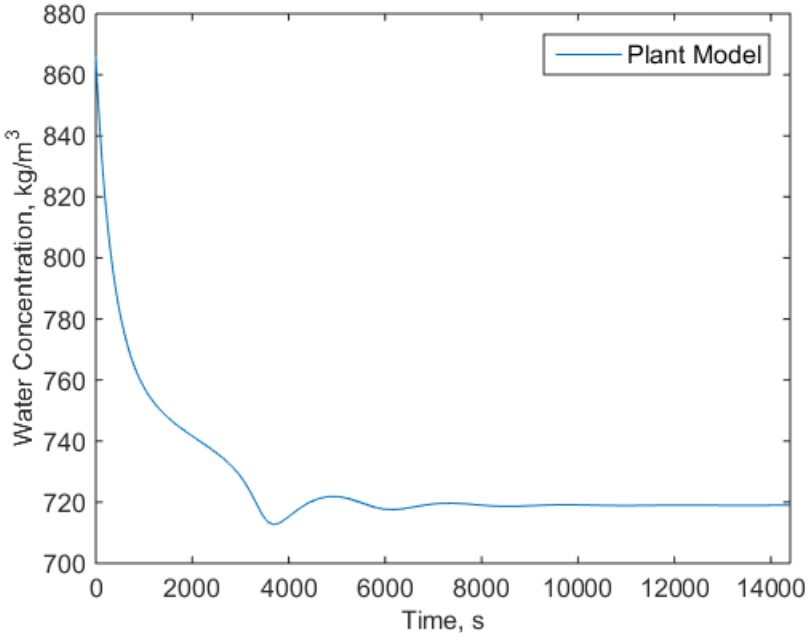

Figure 4.7 Water concentration versus time for propylene glycol reaction simulation model results.

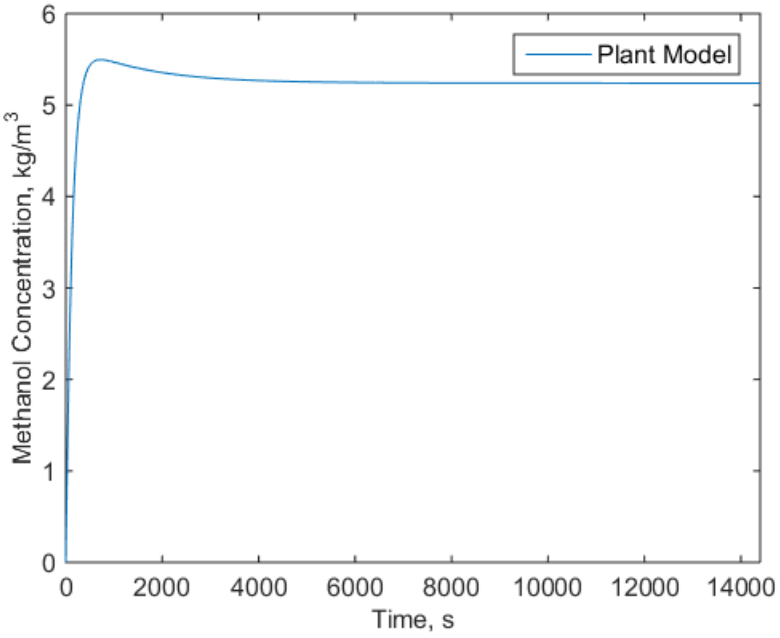

Figure 4.8 Methanol concentration versus time for propylene glycol reaction simulation model results.

4.2 Model Dynamics

An important feature of this modeling toolkit is that interesting and important transient dynamics can be captured. Two examples of dynamic behavior that might be of interest and be undetected with steady state models is the appearance of limit cycles and the amount of temperature overshoot. Limit cycles are important to identify because they can represent potentially damaging system operation and should be avoided.

4.2.1 Limit Cycle Behavior

For a chemical reaction, the equation of the heat of reaction is given by

$$\Delta H_{RX}(T) = \Delta H_R°(T_R) + \Delta c_p(T - T_R), \tag{4.2}$$

where $\Delta H_{RX}(\mathrm{T})$ is the heat of reaction at temperature T, $\Delta H_R°(T_R)$ is heat of reaction at a reference temperature, Δc_p is the overall change in heat capacity, T is the temperature, and T_R

is the reference temperature which is $T_R = 293K = 20C$ for this model [18]. However, sometimes for more simplified models, the heat of reaction is assumed to take the form

$$\Delta H_{RX}(T) = c, \tag{4.3}$$

where c is a constant value.

For the propylene glycol model, the textbook assumes the simplified equation for the heat of reaction of Equation (4.3). The results for varying the heat transfer coefficient between the CSTR jacket and CSTR tank with the simplified equation are rather unremarkable as shown in Figure 4.9. The variation shows increased damping as the heat transfer coefficient increases as expected since there is a coolant flowing through the CSTR jacket. However, if the heat of reaction equation takes the form of Equation (4.2), and is therefore not simplified, more interesting dynamic behavior can occur. Figure 4.10 shows how varying the heat transfer coefficient between the CSTR jacket and CSTR tank can result in limit cycle behavior. A plot of the heat of reaction values is provided in Figure 4.11, showing oscillatory behavior when Equation (4.2) is used versus the assumed constant value for the simplification made in Equation (4.3), which is an assumption that the textbook model makes.

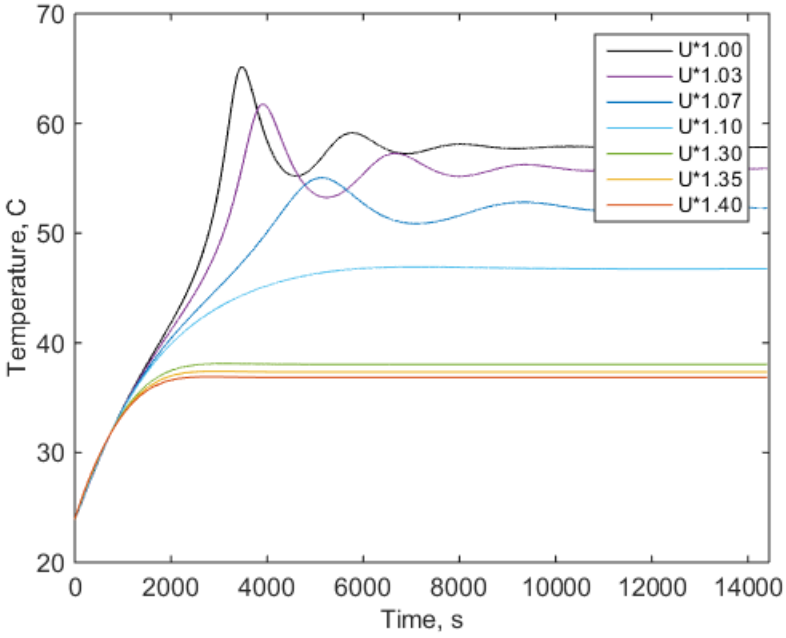

Figure 4.9 Temperature versus time for varying heat transfer coefficients showing how limit cycle behavior does not occur when the heat of reaction equation is simplified according to [37]**.**

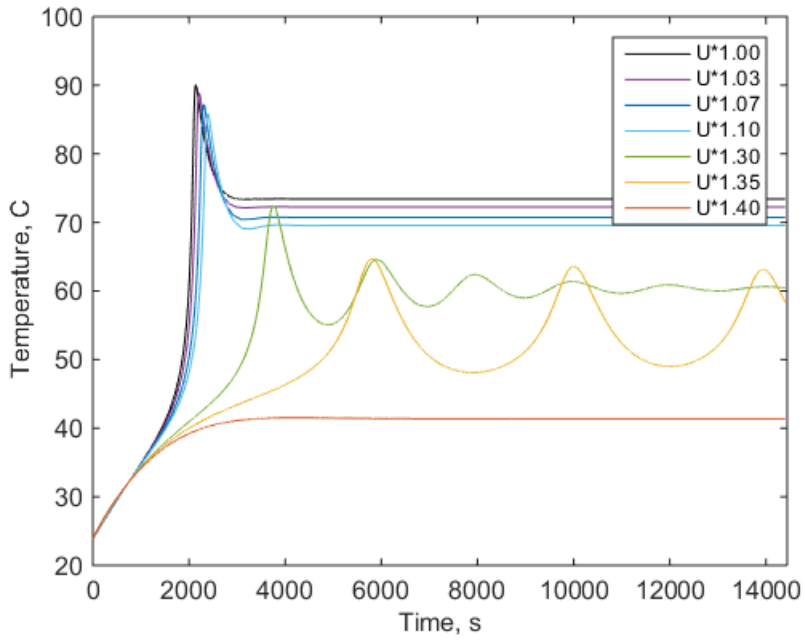

Figure 4.10 Temperature versus time for varying heat transfer coefficients showing how limit cycle behavior appears when the heat of reaction equation is not simplified.

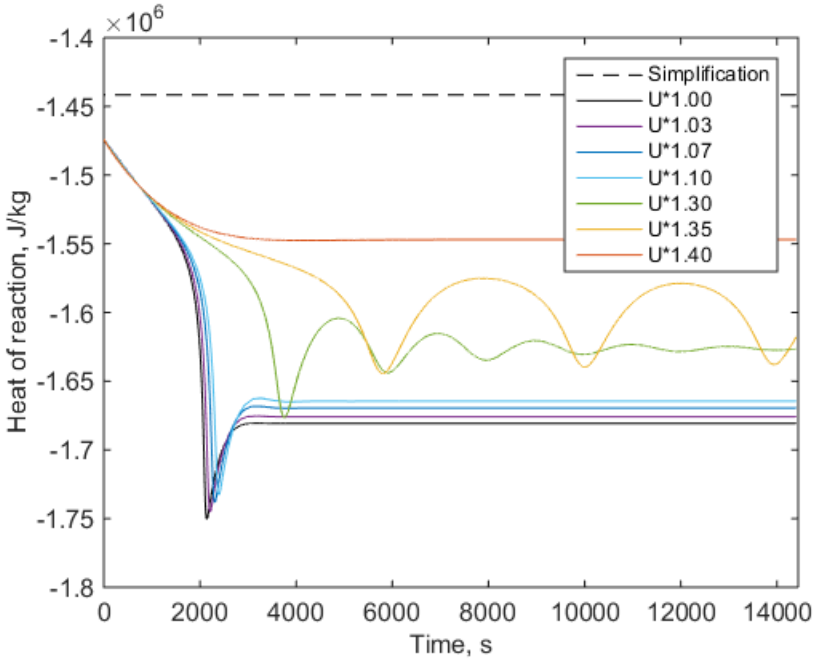

Figure 4.11 Heat of reaction value versus time when the heat of reaction equation is simplified and not simplified for varying heat transfer coefficients.

4.2.2 Temperature Overshoot

Another key aspect of this modeling toolkit is that it can capture useful transient behavior information, such as the reaction temperature overshoot. Figure 4.12 shows how there is a distinct change in how the reaction temperature approaches the steady state value for various inlet mass flow rates of propylene oxide. When the mass flow rate of the incoming propylene oxide is at the nominal value or slightly more, the behavior resembles the response of a second order system. When the mass flow rate of propylene oxide is slightly lower than the nominal value, the behavior resembles the response of a first order system. In certain cases, knowing whether temperature overshoot for a process would occur and how much the overshoot would be may be critically important. Therefore, this is another example of the benefit of having transient behavior information that would not be available with steady state modeling.

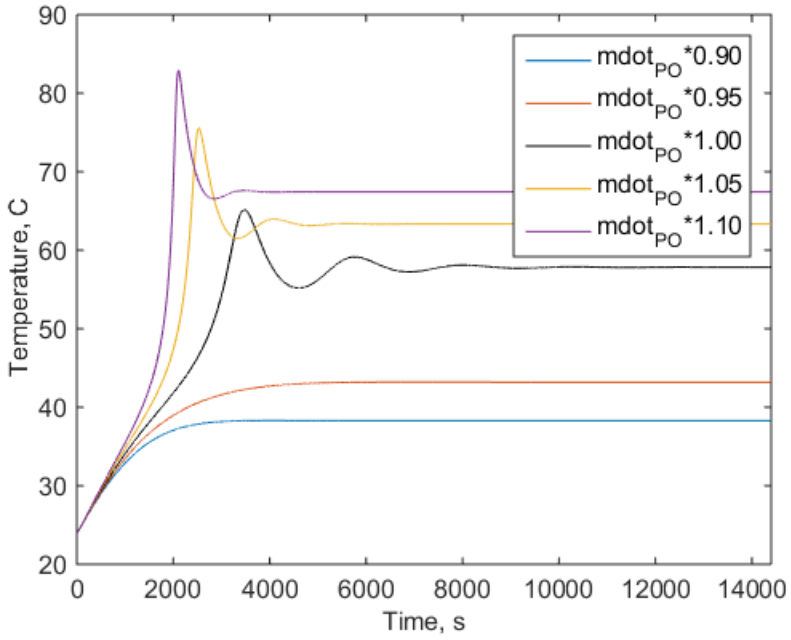

Figure 4.12 Temperature versus time for varying inlet propylene oxide mass flow rates showing different convergence behavior with $mdot_{PO}$ representing the nominal mass flow rate of propylene oxide.

4.3 Model Sensitivity

It is important to note that these models can be quite sensitive to certain inputs and parameters. There are many different ways to conduct formul sensitivity tests. For this research, a formal quantitative sensitivity analysis was not the goal. Instead, the common one-factor-at-a-time (OFAT) approach was taken to better understand qualitatively which inputs and parameters have a relatively large change on the temperature of the reaction for a relatively small percent change from the nominal model values [47]. This approach was taken because it could show whether responses were linear or nonlinear and whether there are tipping points that lead to a drastically different output – such as the change in the temperature resembling a first order response rather than a second order response.

The model sensitivity for nine different inputs and parameters was investigated. Specifically, the effect on the temperature response was studied by quantifying the temperature response in terms of overshoot, settling time, and rise time. The overshoot is the percentage overshoot relative to the final value. The settling time is the time it takes for the response to settle within 2% of the final value. The rise time is the time it takes for the response to rise from 10% to 90% of the steady state value.

To plot the results, the normalized output parameter (eg temperature overshoot) is plotted versus the multiplicative factor of the nominal parameter value. The nominal value is the original parameter value used in the model. For example, for the mass flow rate of methanol, the nominal mass flow rate is 0.08 kg/s and flow rates of $0.90*0.08$ kg/s, $0.99*0.08$ kg/s, $1.00*0.08$ kg/s, $1.01*0.08$ kg/s, and $1.10*0.08$ kg/s are shown in Figure 4.13 as the five red diamonds located at $x = 0.90, 0.99, 1.00, 1.01, 1.10$. The y-axis, the overshoot normalized by nominal, represents the overshoot of each temperature response divided by the overshoot of the temperature response with the nominal methanol mass flow rate of 0.08 kg/s. With this method for plotting the results, all the parameters will have a data point at $(1,1)$ because this is the nominal parameter value and nominal output value that the other parameter values will be compared with.

To assist in visualizing the effect a parameter has on the output, shaded regions are included. The green shaded region represents mild change to the output response for changes to the parameter. The border between the green and yellow regions represents exactly a linear

change (positive or negative linear response). The yellow region represents some change in the output response for changes to the parameter. The border between the yellow and red region represents exactly ten times a linear response or one tenth times a linear response. The red region represents a large or fast change in the output response for small changes to the parameter. The following figures all take the temperature response to be the output considered.

Figure 4.13 shows how sensitive the temperature overshoot is to changes in the inlet water mass flow rate $mdot_W$, the inlet methanol mass flow rate $mdot_M$, and the CSTR jacket heat transfer coefficient U. From Figure 4.13, it can be seen that changes in the inlet methanol mass flow rate, $mdot_M$, have almost no effect on the temperature overshoot since the y-value remains approximately constant and points are within the green region. This behavior can be confirmed by Figure 4.14, which shows the temperature responses for the various mass flow rates all nearly overlapped. The CSTR jacket heat transfer coefficient, U, does seem to have some effect on the overshoot in a negative way with increasing the heat transfer coefficient leading to less overshoot in the yellow region. This behavior can be confirmed by Figure 4.15, which shows the temperature responses for various heat transfer coefficient values. The inlet water mass flow rate, $mdot_W$, causes a large change in the temperature overshoot and falls within the red region. This behavior can be confirmed by Figure 4.16, which shows large changes in behavior for varying inlet water mass flow rates. A useful feature of Figure 4.13 is that it shows when increasing parameters past a certain value (tipping point) results in no overshoot at all, as can be seen with the inlet mass flow rate of water and CSTR jacket heat transfer coefficient with points located at $y = 0$.

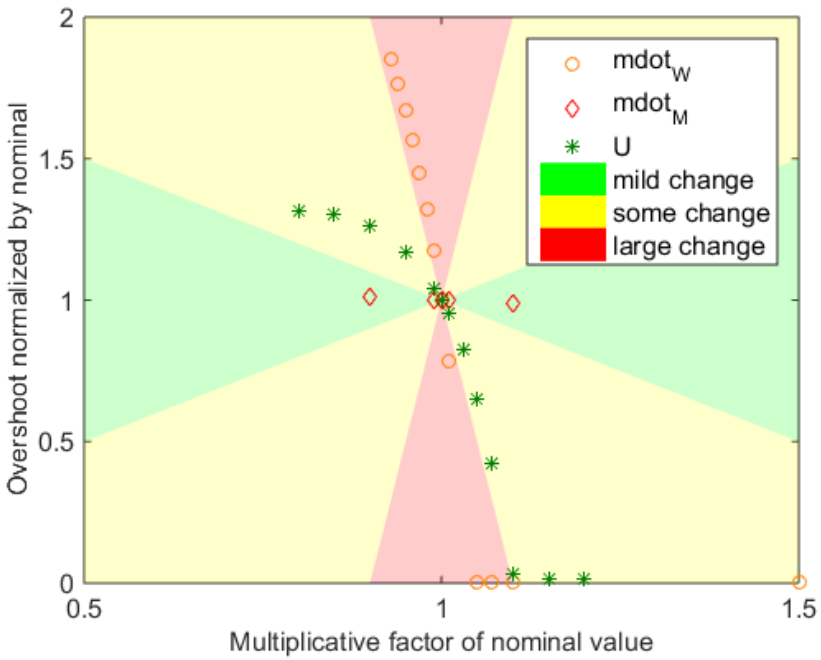

Figure 4.13 Temperature response overshoot versus parameter value for three parameters of interest chosen to demonstrate three distinct sensitivity behaviors.

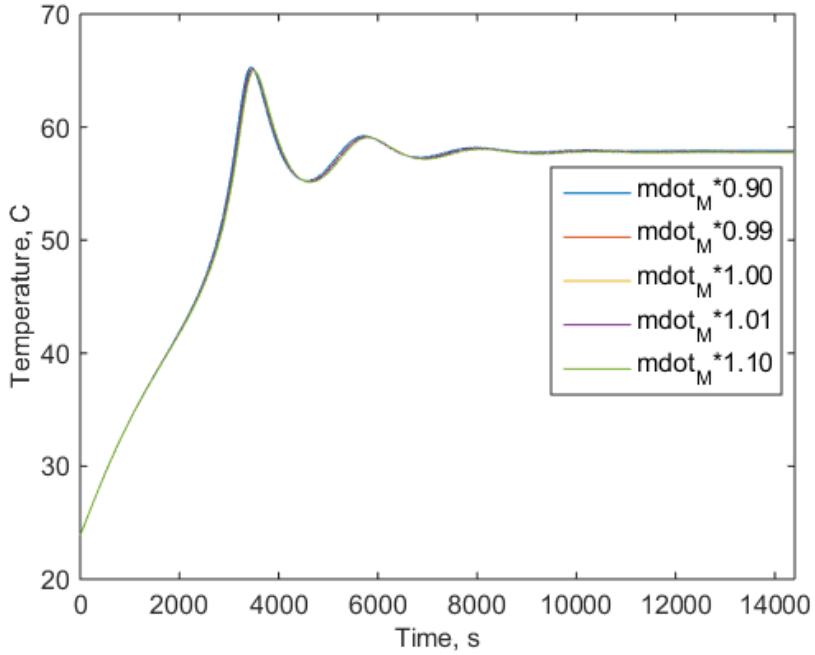

Figure 4.14 Temperature versus time for varying inlet methanol mass flow rates.

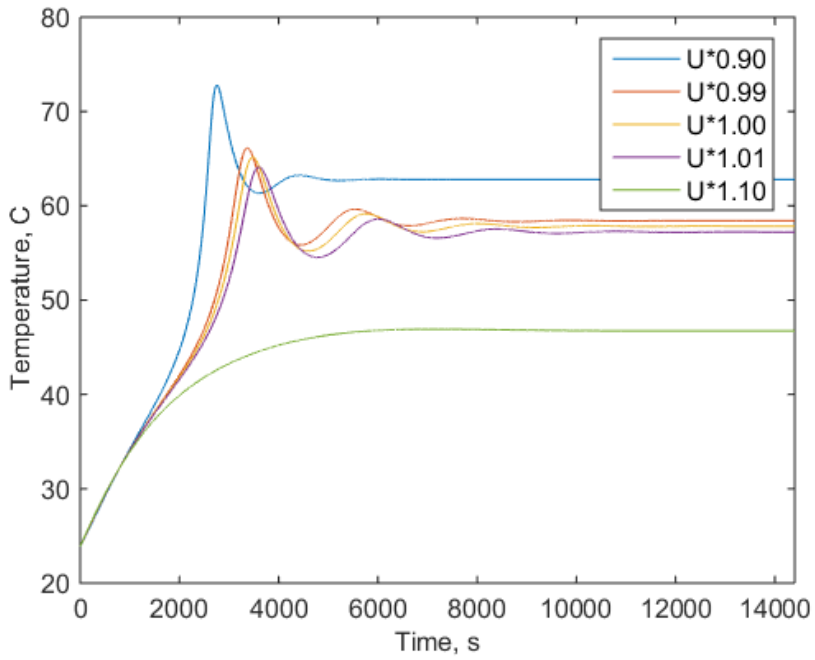

Figure 4.15 Temperature versus time for varying CSTR jacket heat transfer coefficient values.

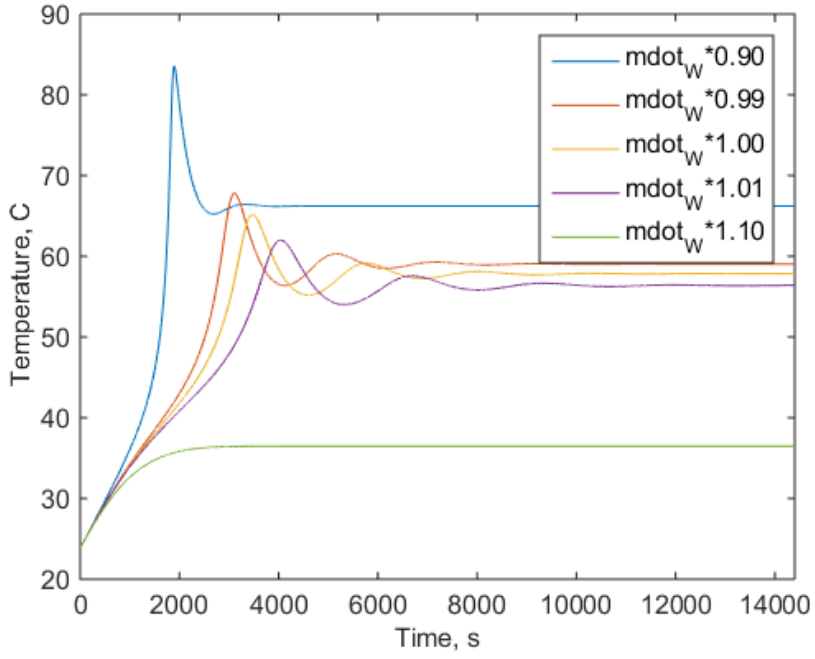

Figure 4.16 Temperature versus time for varying inlet water mass flow rates.

To more clearly visualize what each point Figure 4.13 represents, the next three figures show how the temperature response changes for only varying one parameter – the inlet water mass flow rate. Figure 4.17 has overlaid temperature responses that show how a decreased inlet water mass flow rate leads to much larger overshoot compared to the nominal overshoot with the blue point and its associated plot for 90% of the nominal water mass flow rate. On the other hand, with 110% of the nominal water mass flow rate, the response has switched to having no overshoot at all. To understand the relative scale of the overlaid plots in Figure 4.17, Figure 4.15 can be viewed. Figures 4.18 and 4.19 show with overlaid plots how the settling time and rise time respectively are affected by the change in inlet water mass flow rate.

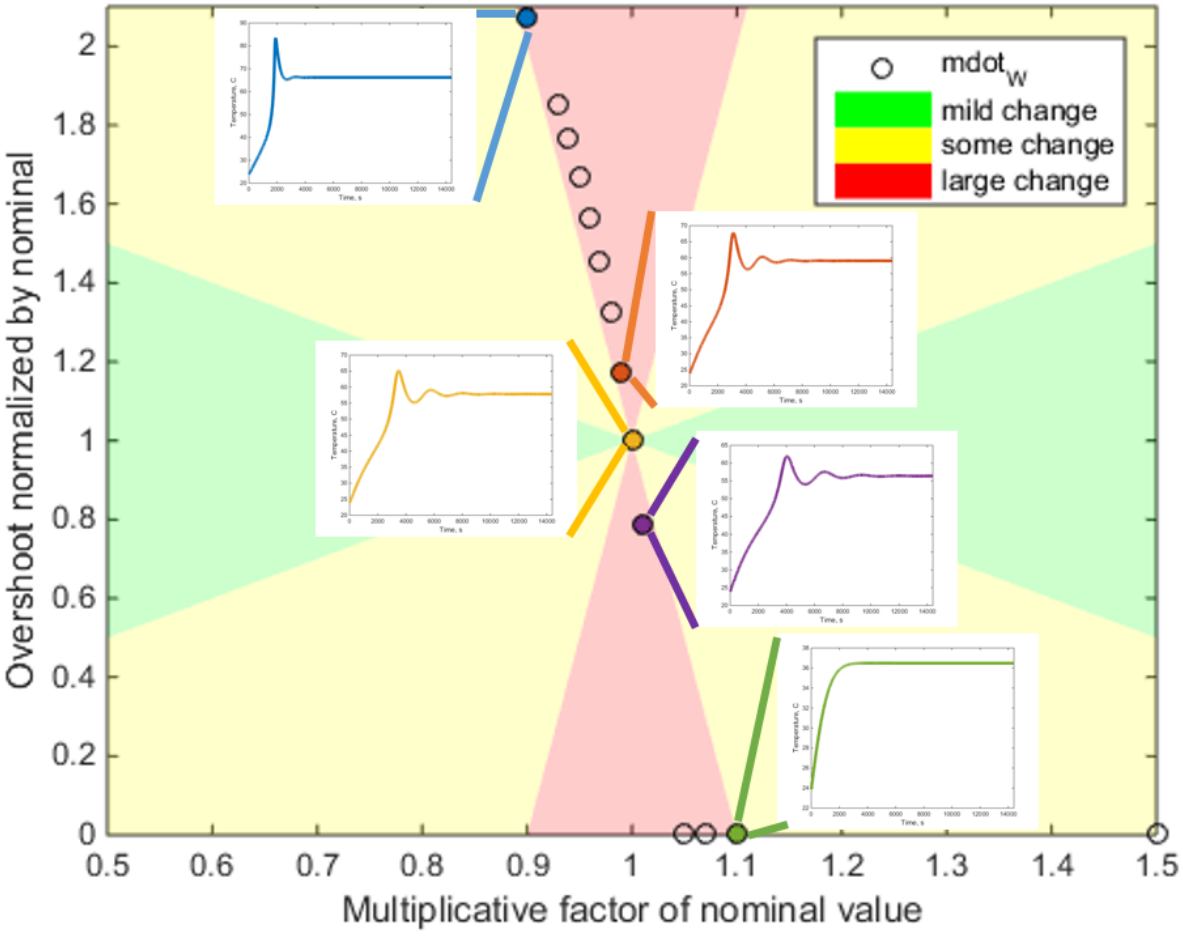

Figure 4.17 Temperature response overshoot versus inlet water mass flow rate with temperature responses for 90%, 99%, 100%, 101%, and 110% of the nominal water mass flow rate overlaid.

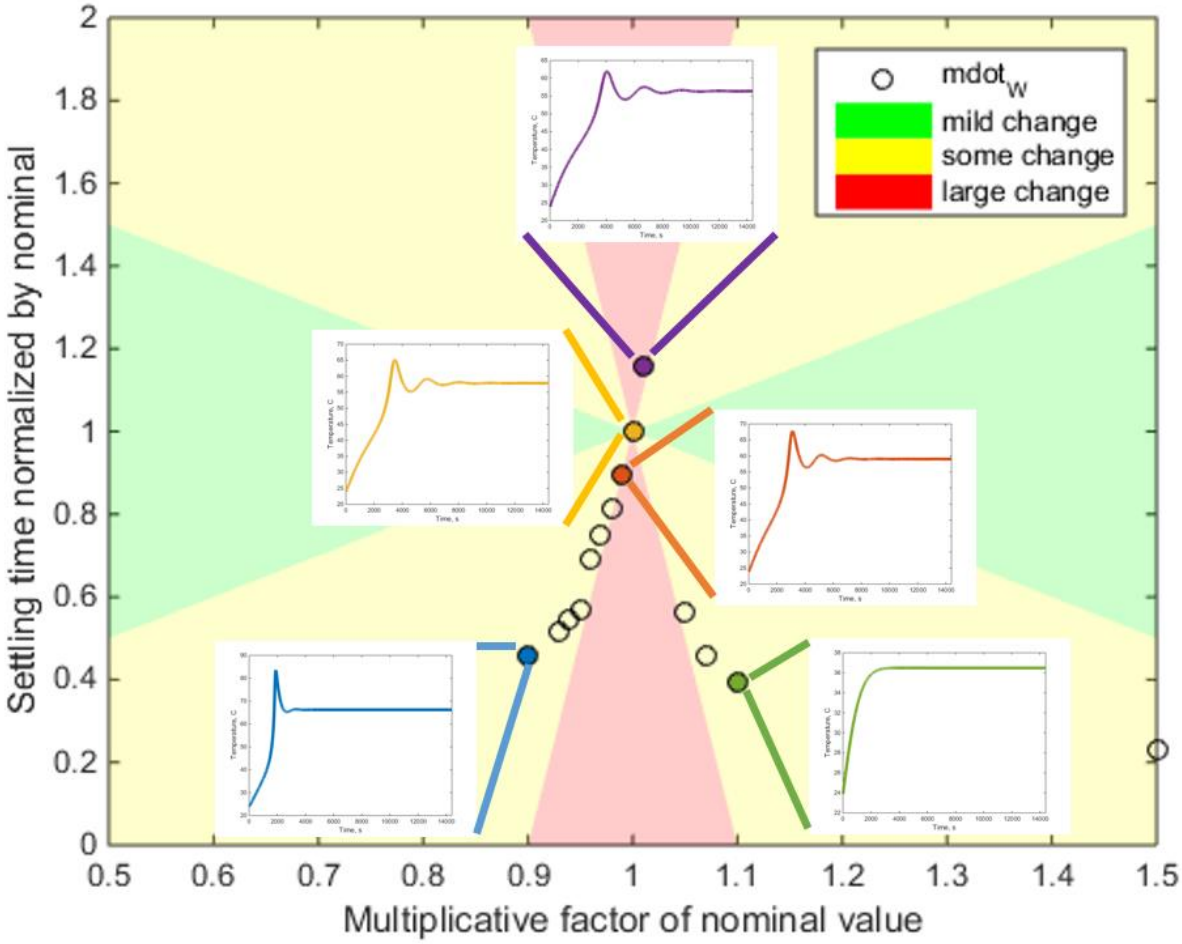

Figure 4.18 Temperature response settling time versus inlet water mass flow rate with temperature responses for 90%, 99%, 100%, 101%, and 110% of the nominal water mass flow rate overlaid.

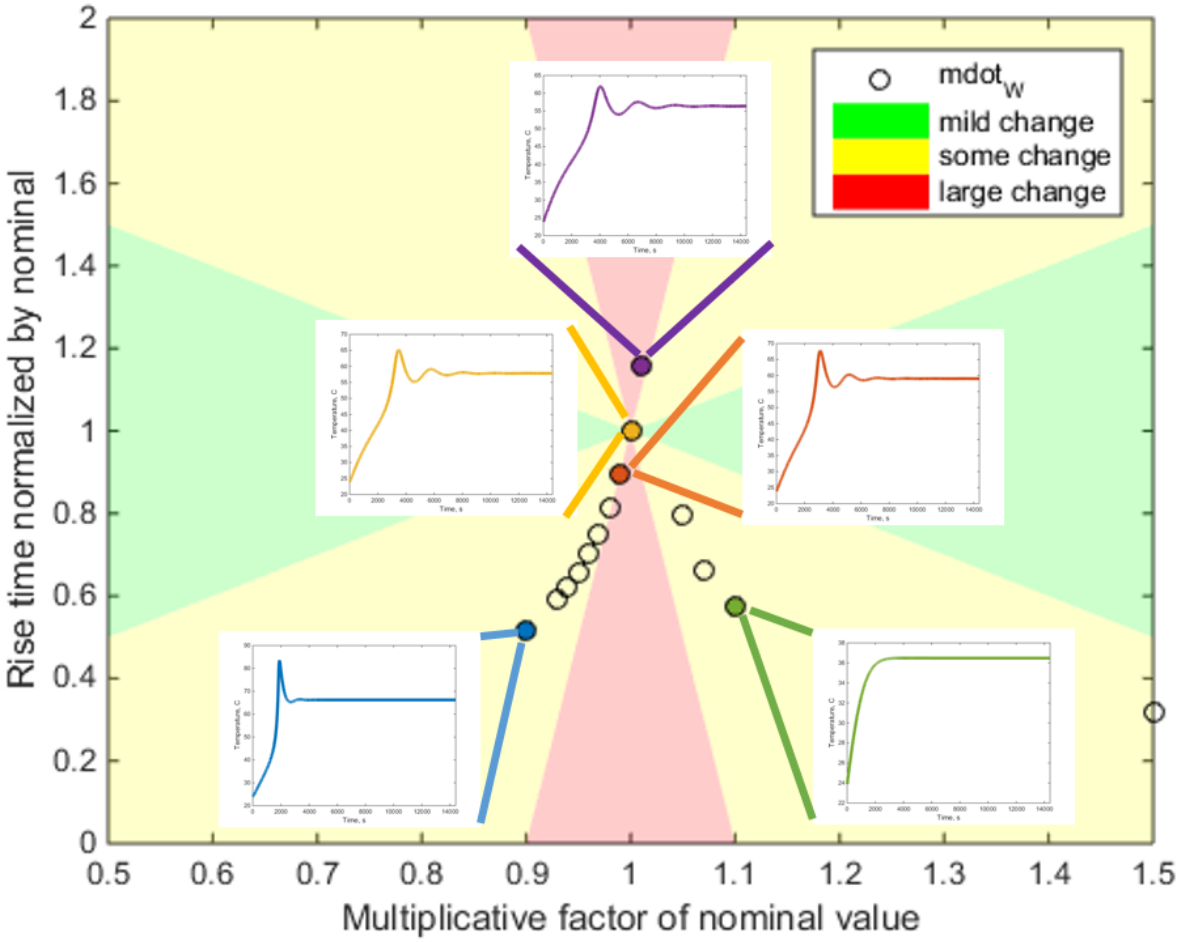

Figure 4.19 Temperature response rise time versus inlet water mass flow rate with temperature responses for 90%, 99%, 100%, 101%, and 110% of the nominal water mass flow rate overlaid.

Using the same convention for the plots, the sensitivity effects of nine different parameters are shown in Figure 4.20, Figure 4.21, and Figure 4.22. The first three parameters are inlet mass flow rates for propylene oxide ($mdot_{PO}$), water ($mdot_{W}$), and methanol ($mdot_{W}$). The next two parameters are geometric parameters where d is the reactor diameter and U is the CSTR jacket heat transfer coefficient. The next two parameters are temperatures where T_{inlet} is the temperature of the all the inlet flows and T_{jacket} is the temperature of the CSTR jacket. The final two parameters are related to the reaction where E is the reaction activation energy and $delta_{HRX}$ is the heat of reaction discussed in Section 4.2.1. These parameters were chosen to

include a wide variety with some as model inputs which can be varied, such as the mass flow rates, some as properties specific to the reaction, and some as plant design parameters, such as the tank diameter.

In summary, the main point of this section is to provide intuition for the relative sensitivity of various parameters. Different metrics could be considered instead and a different output response besides the temperature could be investigated. The intuition for parameter sensitivity is important for constructing models and understanding how changing the heat transfer coefficient for example, influences the amount of overshoot in the temperature response. For instance, if it is desired that the temperature response has no overshoot, then by examining Figure 4.20, it can be understood that the heat transfer coefficient only needs to be slightly greater than the nominal value to have no temperature overshoot at all.

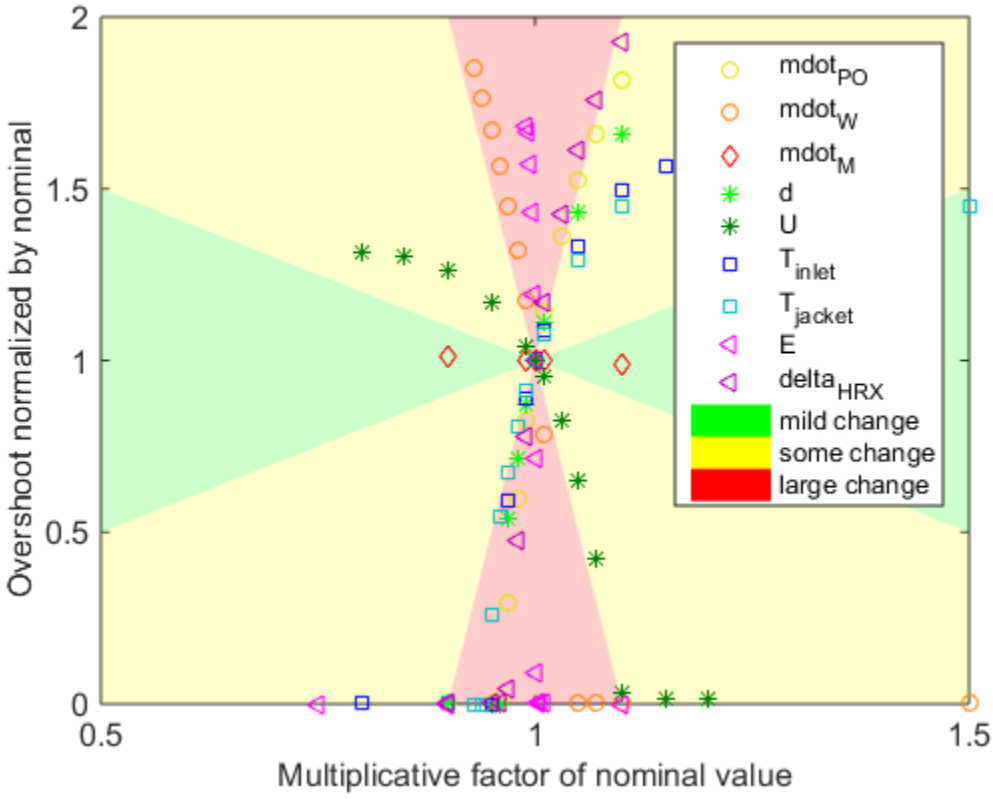

Figure 4.20 Temperature response overshoot versus nine different parameters to show relative sensitivity.

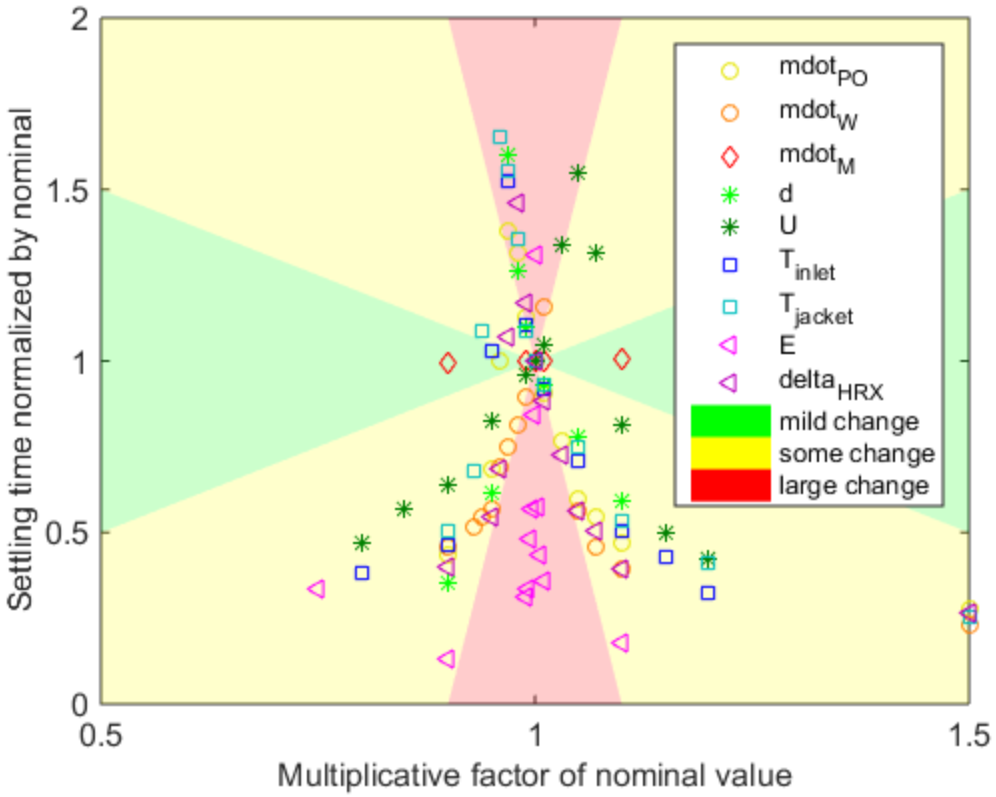

Figure 4.21 Temperature response settling time versus nine different parameters to show relative sensitivity.

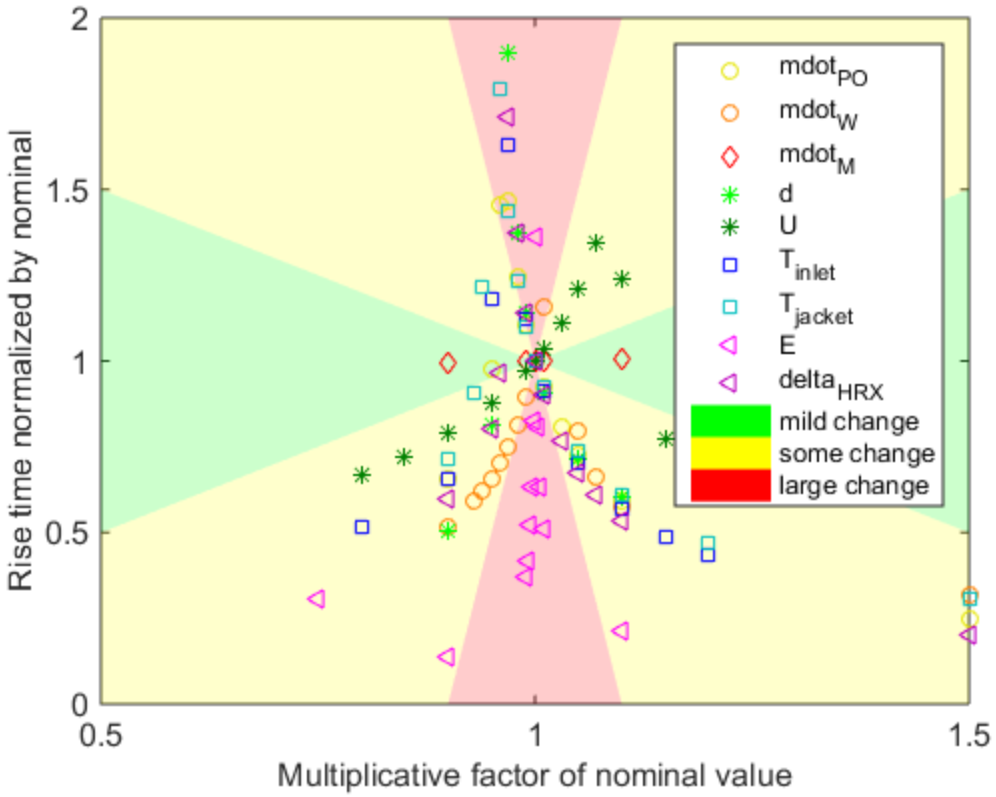

Figure 4.22 Temperature response rise time versus nine different parameters to show relative sensitivity.

4.4 Simulated Control Results

One of the advantages of this modeling toolkit is the ease for which control methods can be applied. The plant model shown in Figure 4.3 will have two simple control schemes applied to demonstrate how easily control can be applied in simulation. The first control scheme includes proportional-integral (PI) control of the CSTR jacket coolant mass flow rate to regulate the temperature in the CSTR. The plant model is the exact same as in Figure 4.3, but now a PI controller determines the coolant mass flow rate instead of having a constant coolant mass flow rate of 1000 kg/s as shown in Figure 4.23. The second control scheme adds PI control of four valves that control the three inlet mass flow rates and the CSTR outlet mass flow rate in addition to PI control of the coolant mass flow rate. Each inlet valve PI control regulates the concentration of its inlet flow and the outlet valve regulates the height of the liquid in the tank.

Figure 4.24 shows how the second control scheme is implemented in Simulink. These PI controllers were not tuned with any particular performance goals, but simply serve to show how easily control can be applied and tested in simulation.

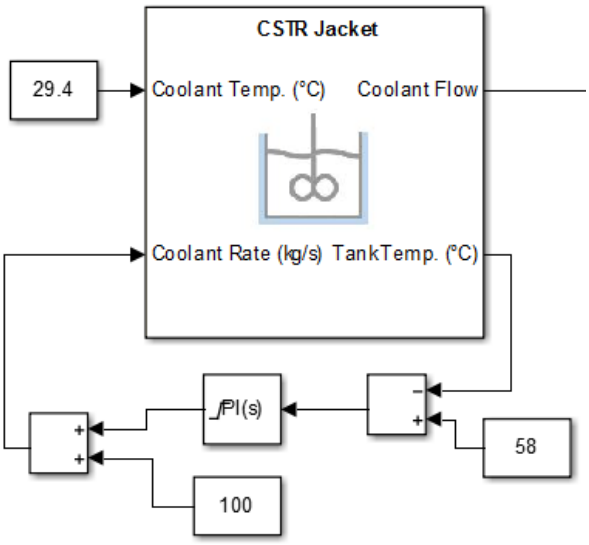

Figure 4.23 First control scheme with only coolant mass flow rate control as applied to the propylene glycol production plant shown in Figure 4.3.

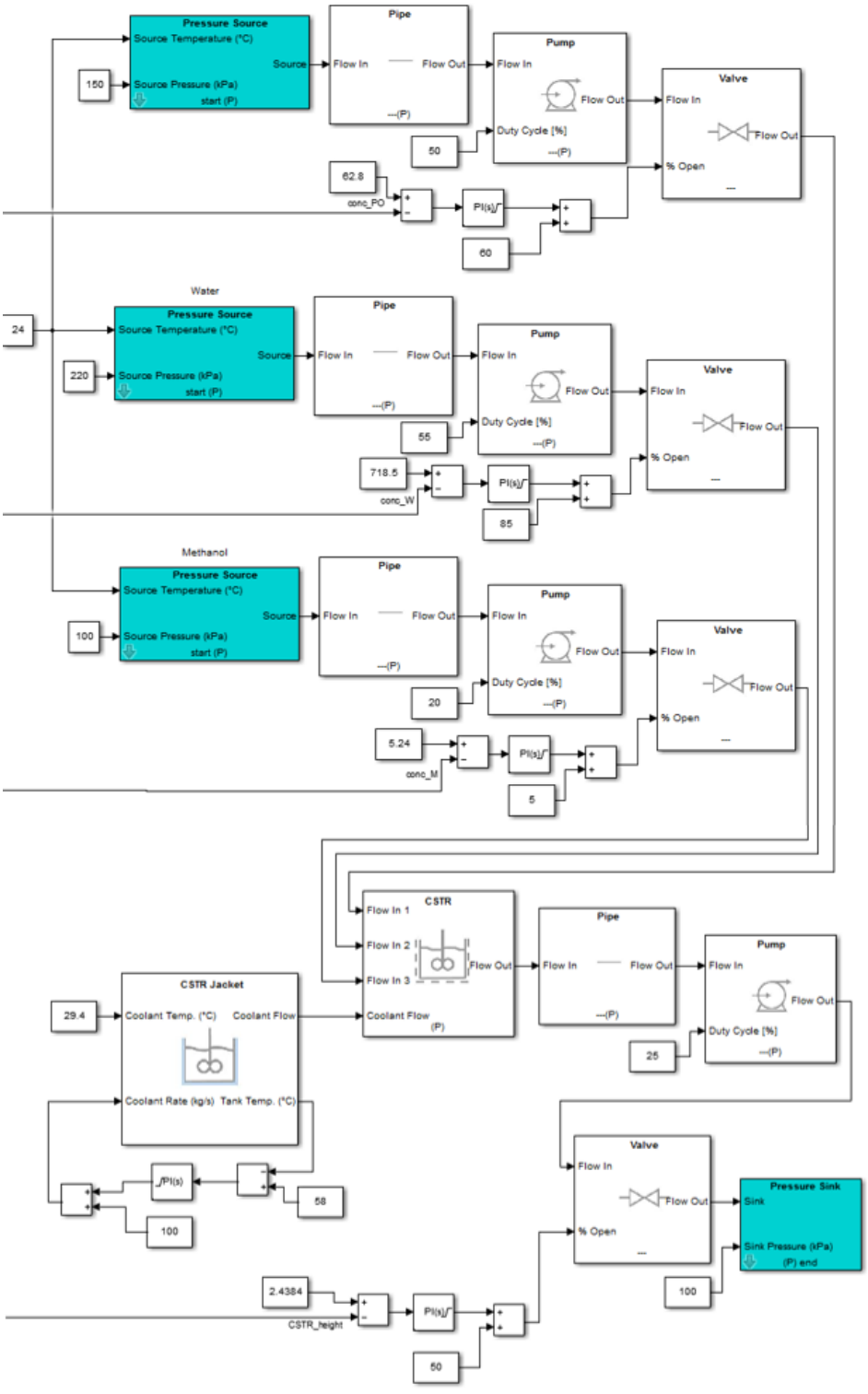

Figure 4.24 Second control scheme with coolant mass flow rate and four valves controlled as applied to the propylene glycol production plant shown in Figure 4.3.

The results of these two control schemes are shown in the following figures. Similar transient and steady state behavior is observed for the temperature and propylene oxide concentration as compared with the textbook results, open loop plant results, and two control schemes as shown in Figures 4.25 and 4.26. The other concentration results are show in Figures 4.27, 4.28, and 4.29. However, it is of interest that the coolant mass flow rate can be lower than the assumed rate of 1000 kg/s in the original open loop plant model as shown in Figure 4.30. Additionally, for the second control scheme, the temperature and concentrations reach their final values faster than with the first control scheme or the open loop model. In other words, the second control scheme leads to a faster settling time, which may be of interest for certain processes. The second control scheme is also more realistic since there likely would be control on the outlet valve to ensure the liquid height in the CSTR remained constant. The valve percent open plots are provided in Figures 4.31, 4.32, 4.33, and 4.34. These plots show how the controlled valve percent open compares to the constant percent open set point for the open loop plant model and when only the coolant mass flow rate is controlled.

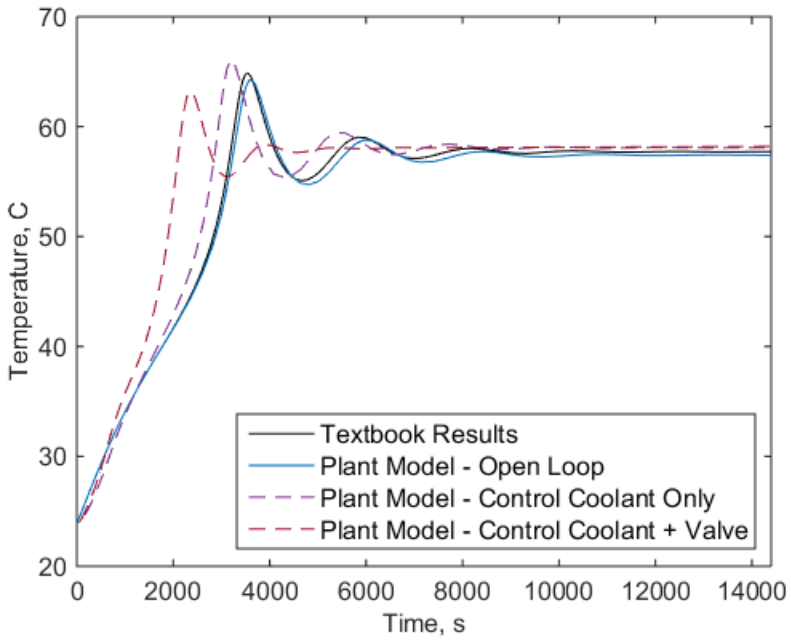

Figure 4.25 Temperature versus time for the textbook results, open loop plant model, and two control schemes implemented on the plant model.

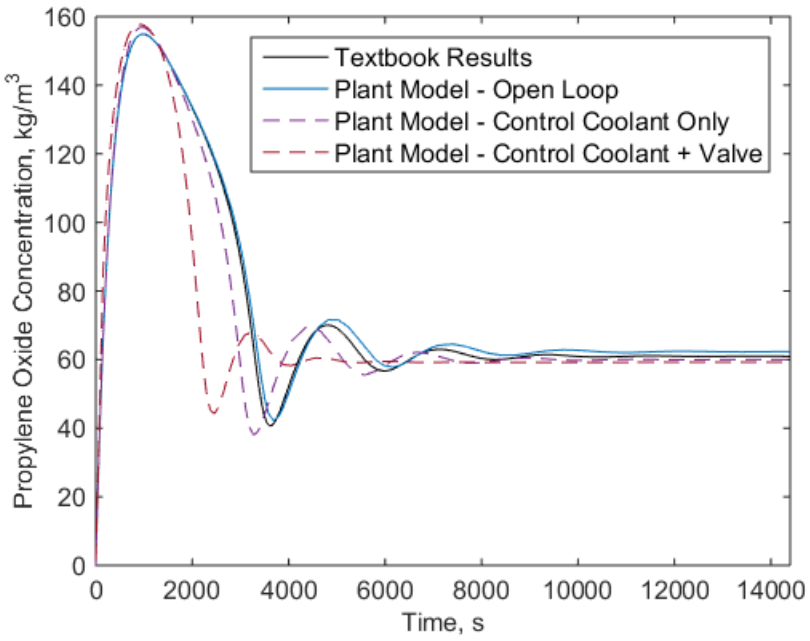

Figure 4.26 Propylene oxide concentration versus time for the textbook results, open loop plant model, and two control schemes implemented on the plant model.

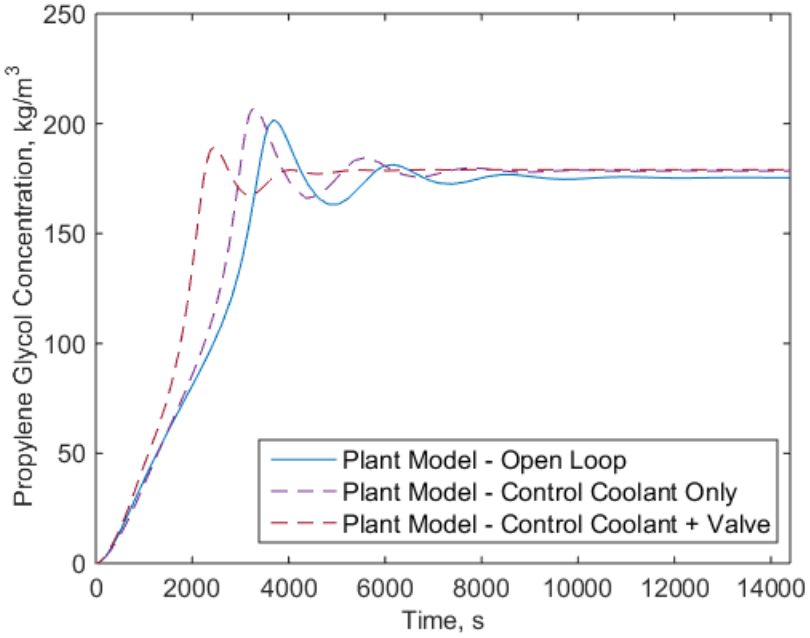

Figure 4.27 Propylene glycol concentration versus time for the open loop plant model and two control schemes implemented on the plant model.

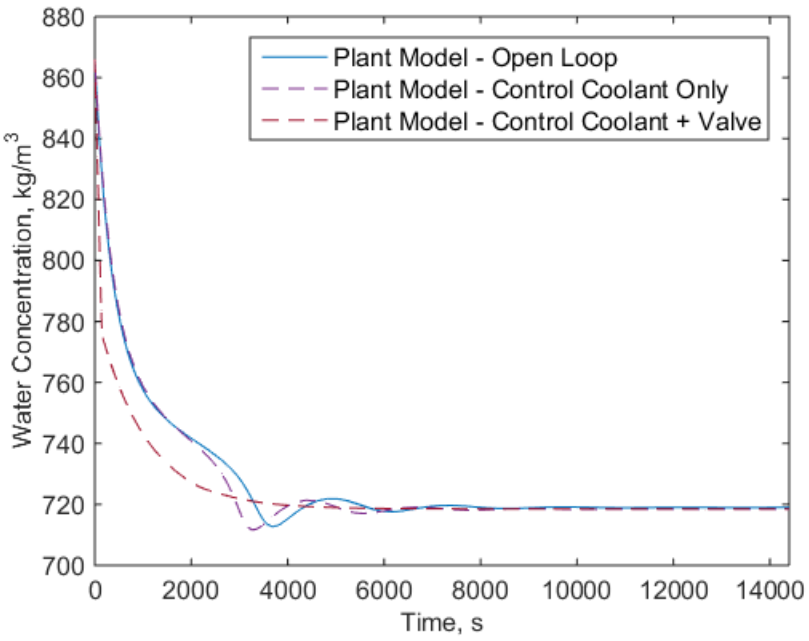

Figure 4.28 Water concentration versus time for the open loop plant model and two control schemes implemented on the plant model.

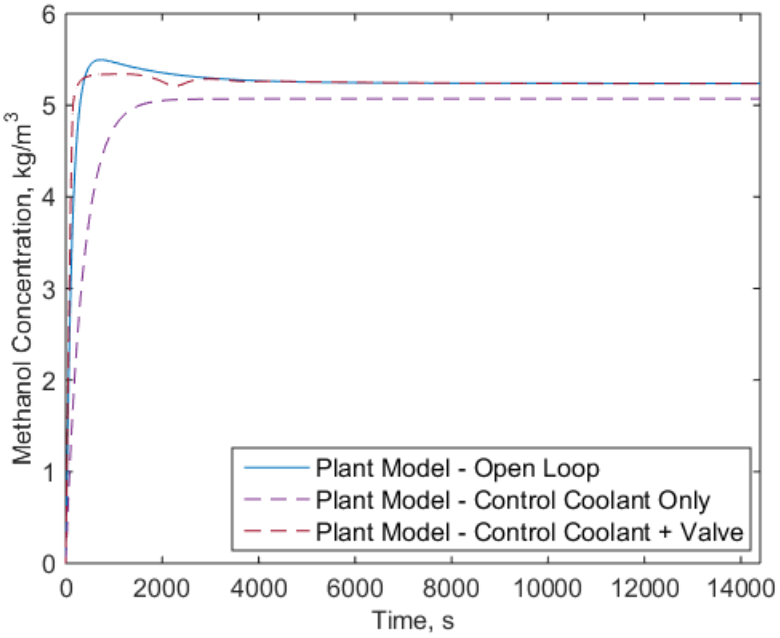

Figure 4.29 Methanol concentration versus time for the open loop plant model and two control schemes implemented on the plant model.

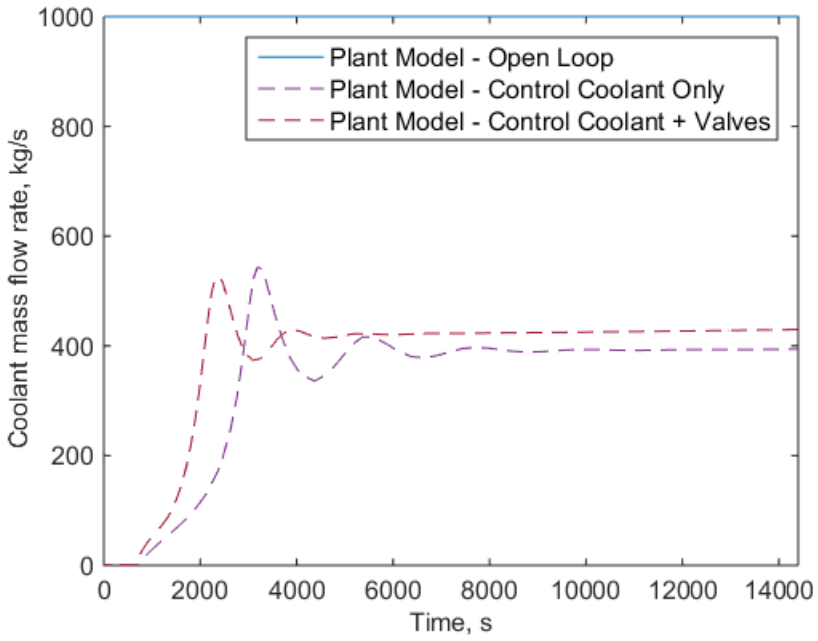

Figure 4.30 Coolant mass flow rate into CSTR jacket versus time for the open loop plant model and two control schemes implemented on the plant model.

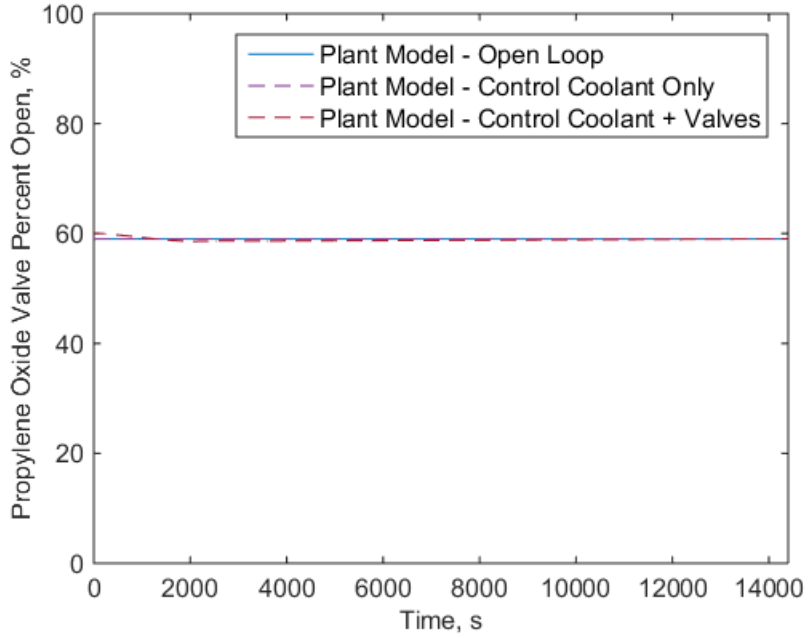

Figure 4.31 Percent open of propylene oxide inlet valve versus time for the open loop plant model and two control schemes implemented on the plant model.

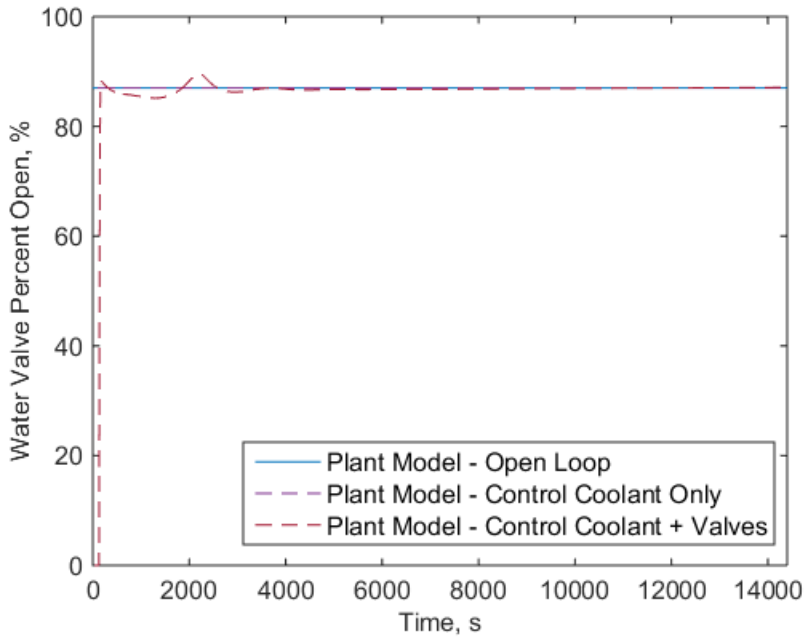

Figure 4.32 Percent open of water inlet valve versus time for the open loop plant model and two control schemes implemented on the plant model.

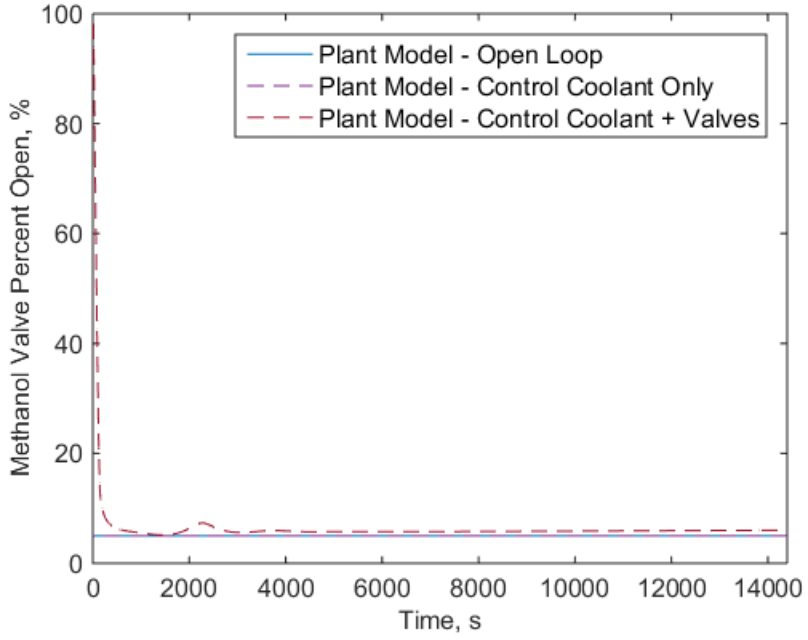

Figure 4.33 Percent open of methanol inlet valve versus time for the open loop plant model and two control schemes implemented on the plant model.

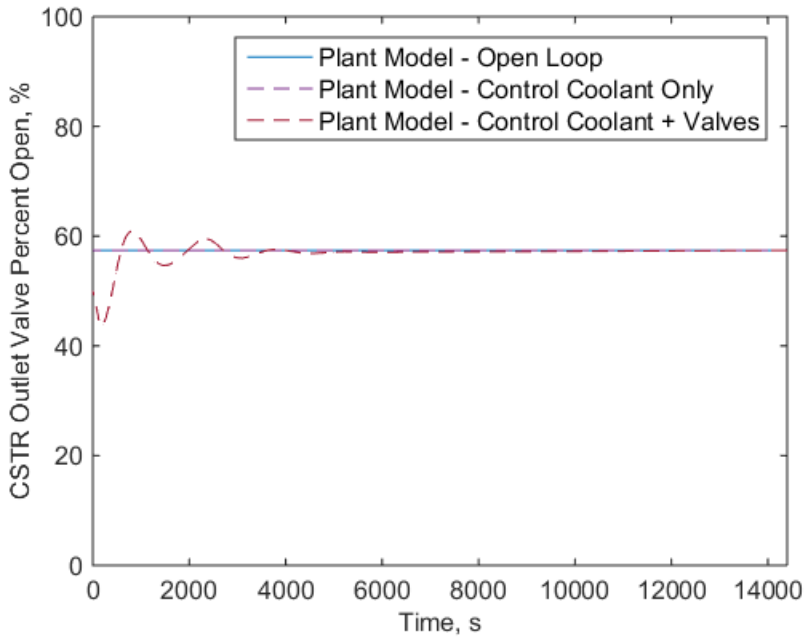

Figure 4.34 Percent open of the outlet valve versus time for the open loop plant model and two control schemes implemented on the plant model.

For the PI control, the built in Simulink PID controllers are used. The anti-windup algorithm based on back-calculation and saturation is utilized [48]. The PI control algorithm used is shown in Figure 4.35. The saturation term is used to ensure that the valves do not exceed percent open values beyond 0 or 100 and the coolant mass flow rate is positive. The controller gains and saturation limits used are provided in Table 4.4.

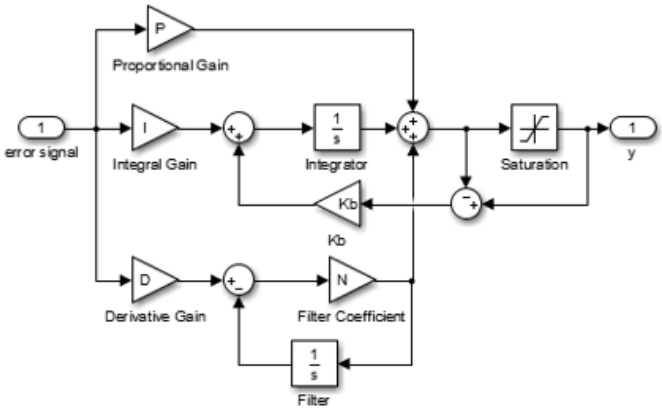

Figure 4.35 PI control anti-windup based on back calculation and saturation algorithm applied [48]**.**

Table 4.4 Controller gains and saturation limits applied in Figure 4.23 and Figure 4.24.

Controller	Proportional Gain	Integral Gain	Kb	Saturation Limits
Coolant rate	-20	-0.005	0.005	2-1000
Valve, Propylene Oxide Inlet	0.001	0.0001	0.0001	0-100
Valve, Water Inlet	10	0.01	0.01	0-100
Valve, Methanol Inlet	20	0.01	0.01	0-100
Valve, CSTR outlet	-10	-0.1	0.01	0-100

Chapter 5
Hardware-in-the-Loop (HIL)

With a dynamic modeling toolkit created (Chapter 2), validation complete (Chapter 3), and process control example developed (Chapter 4), the necessary modeling development towards implementing HIL testing is complete. A dynamic model needed to be developed because, for this HIL implementation, the chemical plant will be emulated. With the plant model complete, the next steps in HIL testing include determining how to run the emulated plant in real-time and what controller will be used. The motivation for HIL testing and details about how the HIL testing was formulated and implemented are included here.

5.1 HIL Advantages in the Chemical Industry

The chemical process industry can benefit from using HIL testing to reduce cost, decrease development time, and improve safety. As explained in Chapter 1, many other industries, especially the automotive industry, are already using HIL testing extensively and seeing benefits in time and cost savings [4], [6], [7]. The current chemical process modeling techniques explored in Chapter 2 include Aspen HYSYS or Matlab Simulink primarily. The standard hardware used for control are PLCs and typical control techniques focus around steady state regulatory set points for PID control. Therefore, it is clear that the HIL development provides the ability to more closely bridge the gap between dynamic modeling and the physical hardware that is used to implement the desired process control algorithms. The widespread use of HIL testing in other industries implies how instrumental HIL development can be. Many of the benefits of HIL testing in other industries are already being realized in the chemical process

industry, but there remains a significant gap that can be filled to optimize chemical processes [10], [12], [13], [15].

5.2 Hardware and Software Selected

5.2.1 Emulated Plant

The plant model developed in Matlab Simulink needs to run in real-time on hardware to accurately emulate a plant. As previously mentioned, it is important to have the emulated plant run in real-time so that the behavior matches the real plant behavior as closely as possible and control algorithms can be accurately tested. A National Instruments CompactRIO (cRIO) was chosen as the hardware to be the emulated plant. A cRIO is a programmable automation controller with embedded real-time/FPGA technology [49]. A cRIO has many capabilities, but the key properties for this research include that it has the capability to run a Simulink model in real-time and has analog and digital I/O modules. A cRIO-9035 is used with added I/O modules. There is an analog input module (9205), analog output module (9264), digital input module (9425), and digital output module (9476). These four modules allow for the emulated plant to send and receive analog and digital signals. Thus, the cRIO serving as the emulated plant can interact with other components, such as a controller for this research. In the future, it would be possible to have pieces of the model not be simulated on the cRIO, but be physical components, and a physical component could then be integrated along with the cRIO through the I/O.

A crucial step to using the cRIO as the emulated real-time plant was converting the Simulink model to a particular form. The software used to do this was National Instruments VeriStand. VeriStand is a real-time testing environment designed specifically for HIL testing and most importantly for this research, provides simulation model integration [50]. Additionally, VeriStand provides a user interface, which allows model states and cRIO I/O to be monitored. Figure 5.1 depicts how VeriStand is the link between having a plant model running on a desktop computer in Matlab Simulink and having an emulated real-time plant running on a cRIO with I/O. Detailed instructions for converting a Simulink model using VeriStand to run on a cRIO follow.

Figure 5.1 Schematic showing how NI VeriStand is the link between the plant model in Matlab Simulink running on a computer and having an emulated real-time plant running on a CompactRIO.

5.2.1.1 Steps to Convert Simulink Model Integrate with VeriStand

To run a Simulink model in real-time on a cRIO, the model must first be able to run with a fixed-step solver in Simulink in real-time. Once the model can do this, these steps lead to the development of the proper file type needed to load model in VeriStand to run on a cRIO. This process was completed using Matlab version 2014b and VeriStand 2106. It is of interest to note that VeriStand 2015 was unable to correctly build the Simulink model, but VeriStand 2016 had updates that allowed this procedure to work.

1. Open the Simulink model (*.slx file) that will be run on the cRIO.
2. Determine which model inputs will be controlled on the VeriStand user interface or through the cRIO analog or digital inputs and replace constants with inport blocks as shown with the inport blocks highlighted in yellow in Figure 5.2.
3. Determine which model outputs will be read on the VeriStand user interface or communicated through the cRIO analog or digital outputs and add outport blocks as shown with the outport blocks highlighted in purple in Figure 5.2. Notice that the outport blocks can be direct outputs from main blocks or can originate from the data manager.
4. In the Model Configuration Parameters, edit the Solver settings as shown in Figure 5.3. The stop time should be set to “inf” for infinity. The solver type must be a “Fixed-step” solver instead of a “Variable-step” solver. Choose the desired fixed-step solver. The models for this research are often stiff, making “ode14x (extrapolation)” a good solver choice because it is the only stiff fixed-step

solver. Additionally, select the desired fixed-step size (fundamental sample time).

5. In the Model Configuration Parameters, edit the Code Generation options as shown in Figure 5.4. Click “Browse…” to select the system target file “NIVeriStand_Linux_64.tlc”. It is important to change the target file type from “grt.tlc”, which is the default generic real-time target.
6. Build the model by clicking the build button or ctrl+b. Building the model will create a folder called “filename_niVeriStand_Linux_64_rtw” in same directory as the original model *.slx file. The necessary file in the build generation is the shared object file called “libfilename.so”, where filename is the *.slx filename. Once the *.so file has been generated, Matlab can be closed.

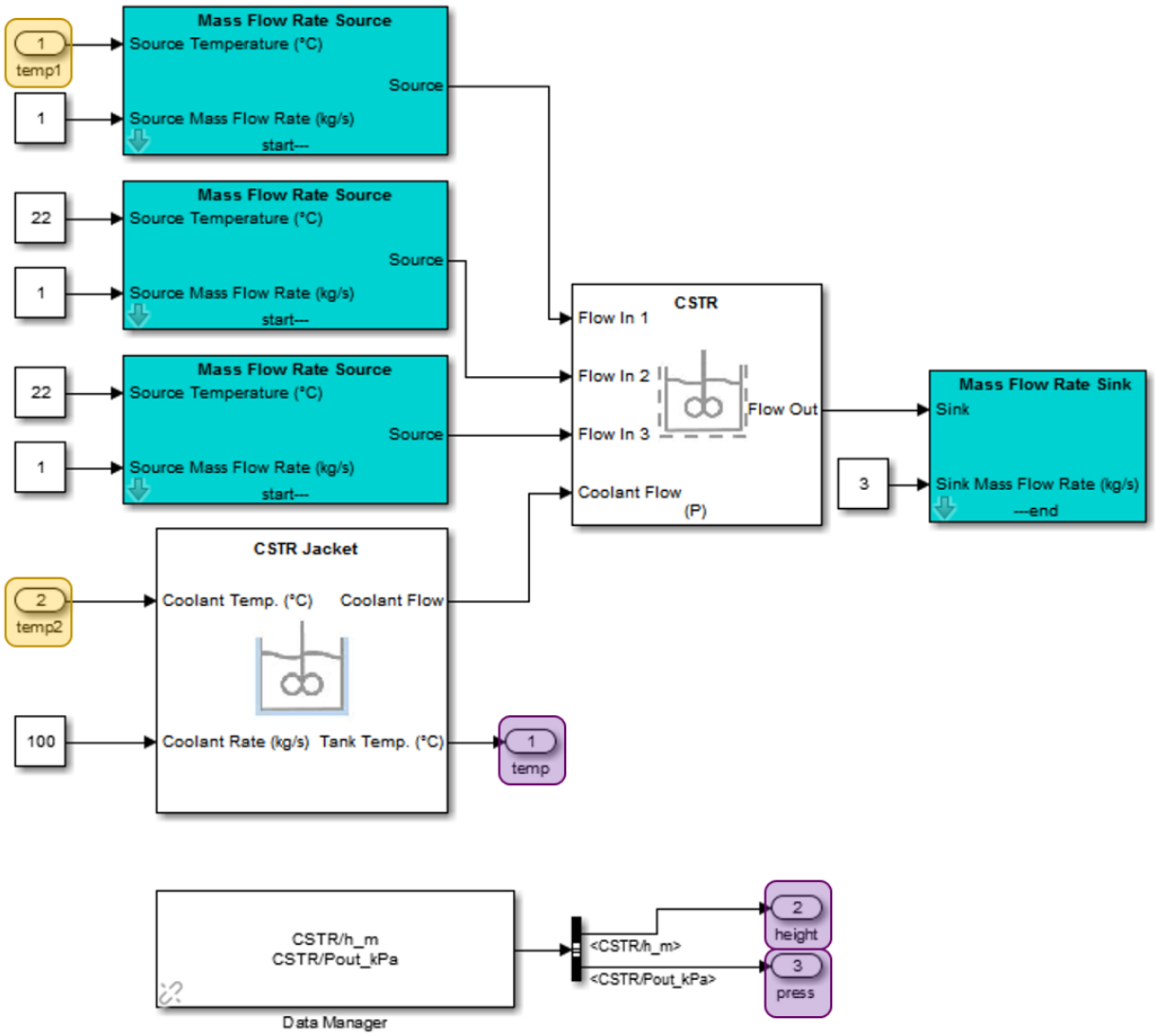

Figure 5.2 Example Simulink model file showing added two inports highlighted in yellow and three outports highlighted in purple.

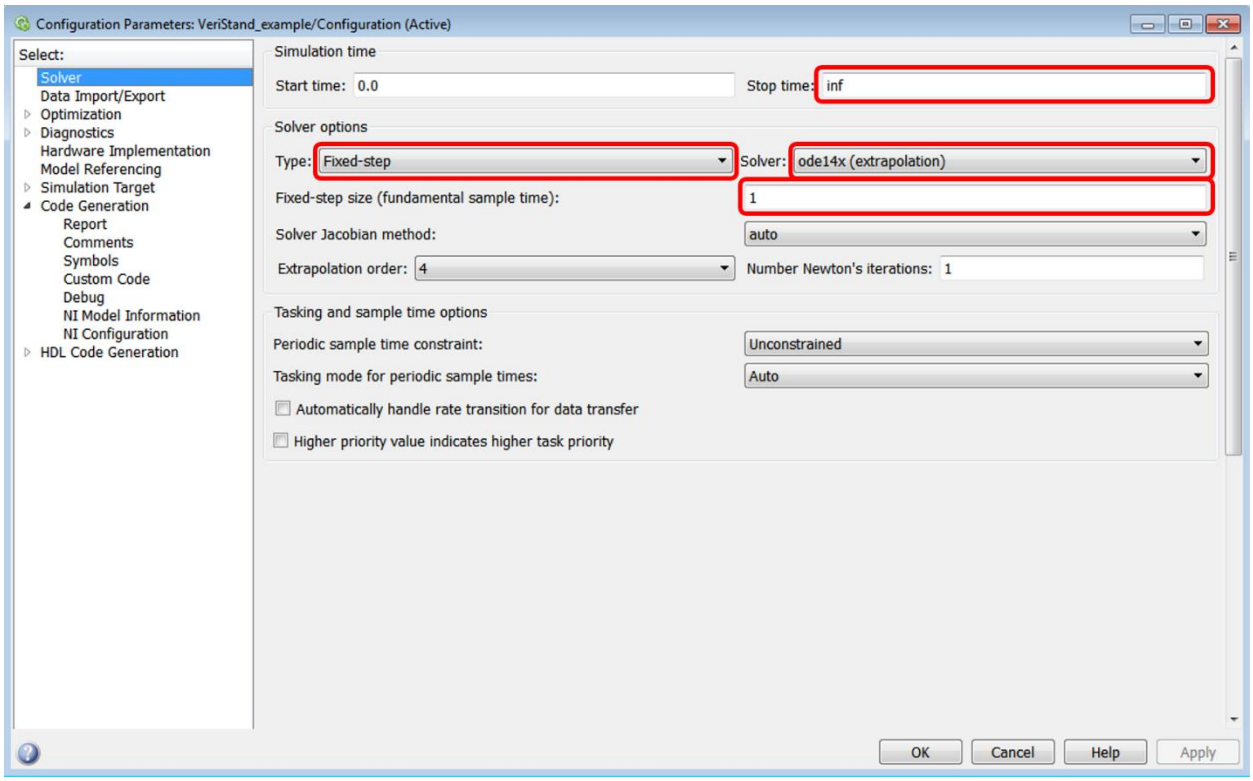

Figure 5.3 Example Simulink Model Configuration Parameter Solver settings with key choices emphasized in red.

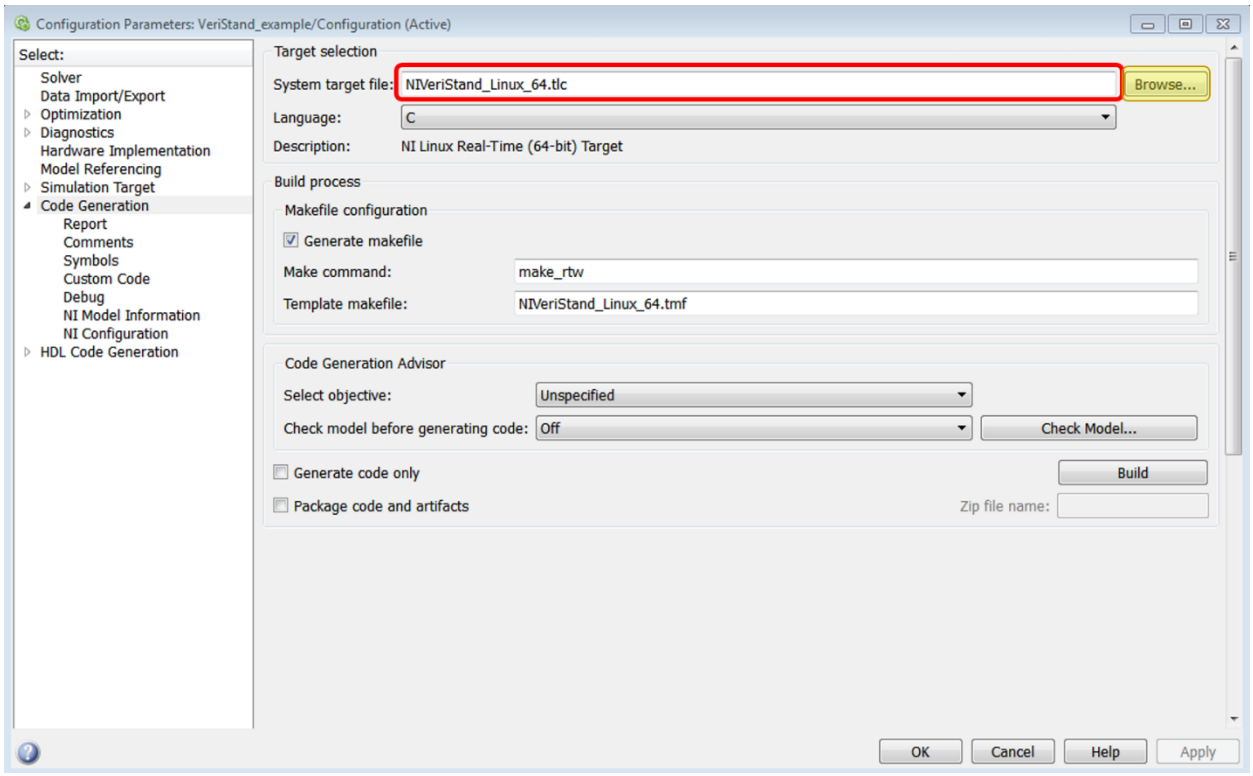

Figure 5.4 Example Simulink Model Configuration Parameter Code Generation options with the desired target file emphasized in red.

5.2.1.2 Steps to Run Model on CompactRIO with VeriStand

After the shared object file from the Simulink model is created, two VeriStand files need to be edited to have the model run in real-time on the cRIO. For this process, the cRIO needs to have power and be connected via Ethernet to the same network switch as the computer VeriStand is operating on, in order to be detected in the method described herein. The appropriate cRIO software needs to be downloaded as well, which in particular needs to include NI VeriStand Engine. The cRIO can be configured for the first time and software downloaded with NI Measurement and Automation Explorer (MAX). For this project, VeriStand 2016 was used. Additionally, the Scan Engine and EtherCAT 2016 v4.3 add-on for VeriStand was needed to connect to the cRIO with VeriStand. After a new VeriStand project is created (*.nivsproj file), these are the steps to edit the System Definition file.

1. Open the System Definition File with the file extension *.nivssdf.

2. Configure the cRIO connection by clicking on Controller as shown in Figure 5.5. Select “Linux_x64” as the operating system. Type in the cRIO IP address (eg 130.126.176.59). Set the Timing Source Settings Target Rate to be the same as the Simulink model rate. For example, to be consistent with the a model step size of 1 second, the Target Rate should be 1 Hz. Leave the Primary Control Loop timing source as Automatic Timing.
3. Add the cRIO as a target by right clicking on Custom Devices -> National Instruments -> Scan Engine & EtherCAT. Next, click the “Auto-Detect Modules” button along the top, then click “Yes” within the dialog box that pops up. The Local Chassis and any additional modules in the cRIO will show up as shown in Figure 5.6.
4. Add the Simulink model by clicking on Simulation Models, then the “Add a Simulation Model” button along the top. Find the desired shared object library file (*.so) created in Simulink and then click OK to add the model. Notice that the model information, including the model rate (which should match the rate set in step 2) is shown once the model is imported. Switch to “Initial state paused” and ensure the decimation is set to 1 as shown in Figure 5.7. One can expand the Inports and Exports folders to see the signals that were set as inports and outports in the Simulink file in Figure 5.2.
5. It is likely that one will need to create custom scales to convert values with physical meaning (eg temperature) to or from a voltage. To create a custom scale, right click on Scales and add a Polynomial Scale. Enter the name and type in the desired coefficients. For the cRIO analog input channel, to convert a voltage (-10V to +10V) to a coolant temperature, for example, one should use the forward coefficients. To convert a value such as the tank temperature being sent to a cRIO analog output channel, one should use the reverse coefficients as shown in Figure 5.8. Notice that both the forward and reverse coefficients are required, but one can use the “Generate” button to auto generate the other direction coefficients.

6. To apply a custom scale to a particular channel, while clicked within the Scales menu, click the “Map Scales” button along the top. Then, select the desired scale (eg temp_to_V) and the channel to be mapped before clicking “Connect” as demonstrated in Figure 5.9.
7. To connect particular cRIO channels with particular model channels, use the “Configure Mappings” button shown in Figure 5.8 and connect desired cRIO I/O and model channels as shown in Figure 5.10.

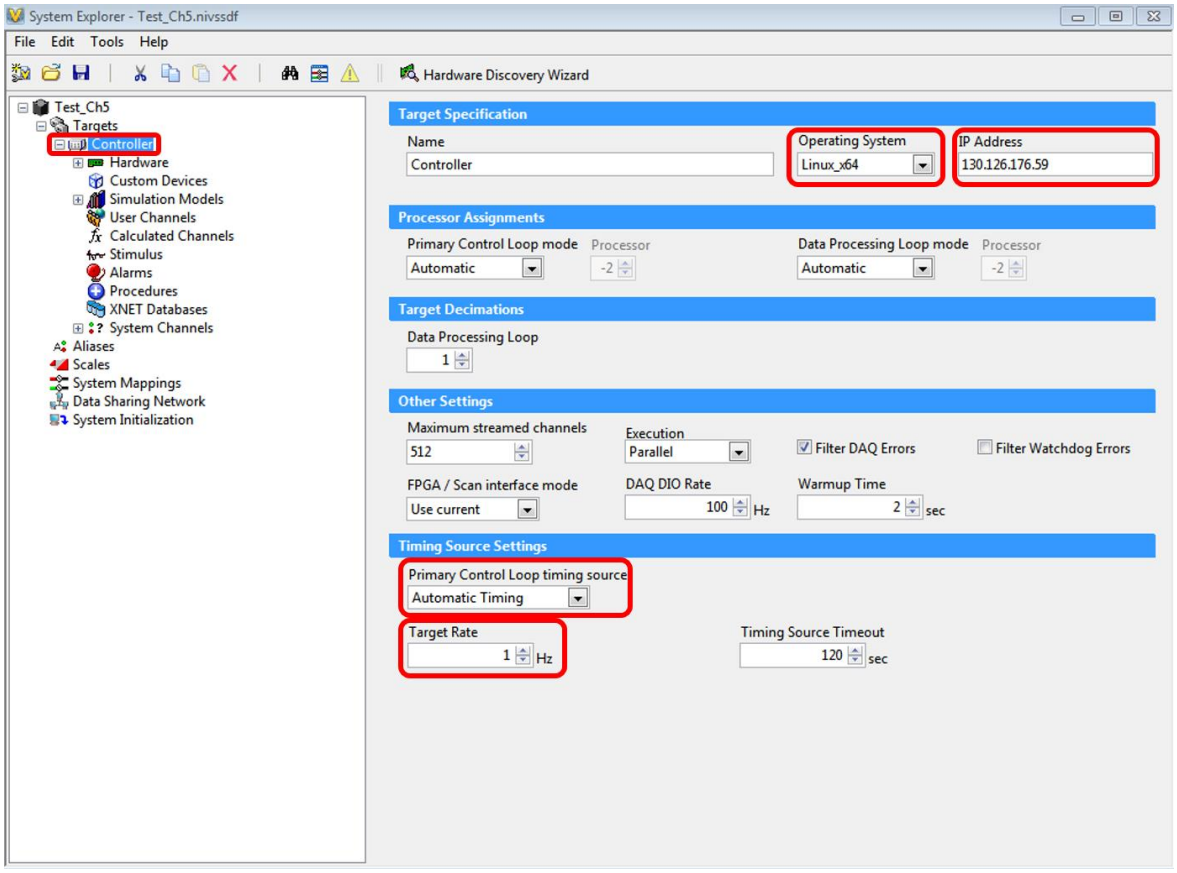

Figure 5.5 Example VeriStand System Definition Controller setup process.

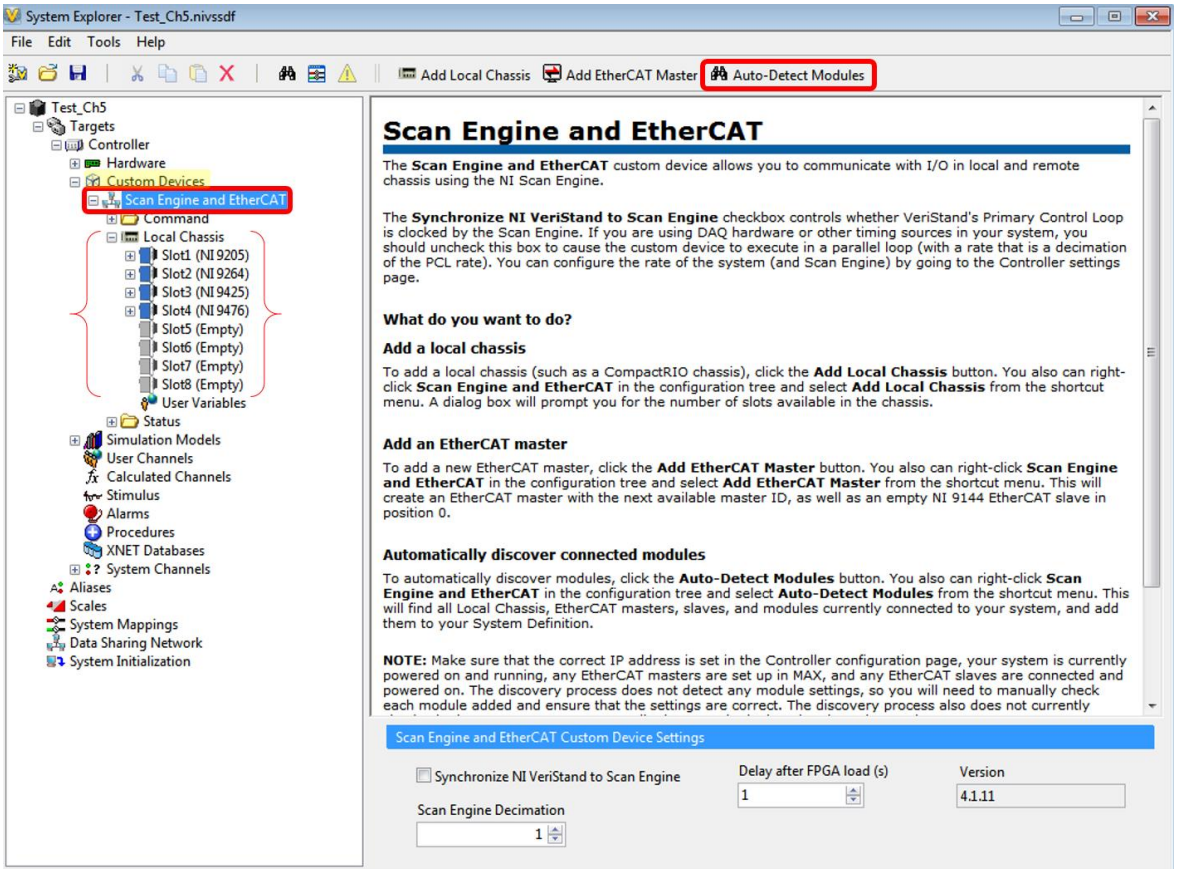

Figure 5.6 Example VeriStand System Definition file cRIO module detection showing the cRIO Local Chassis detected as well as the four I/O modules.

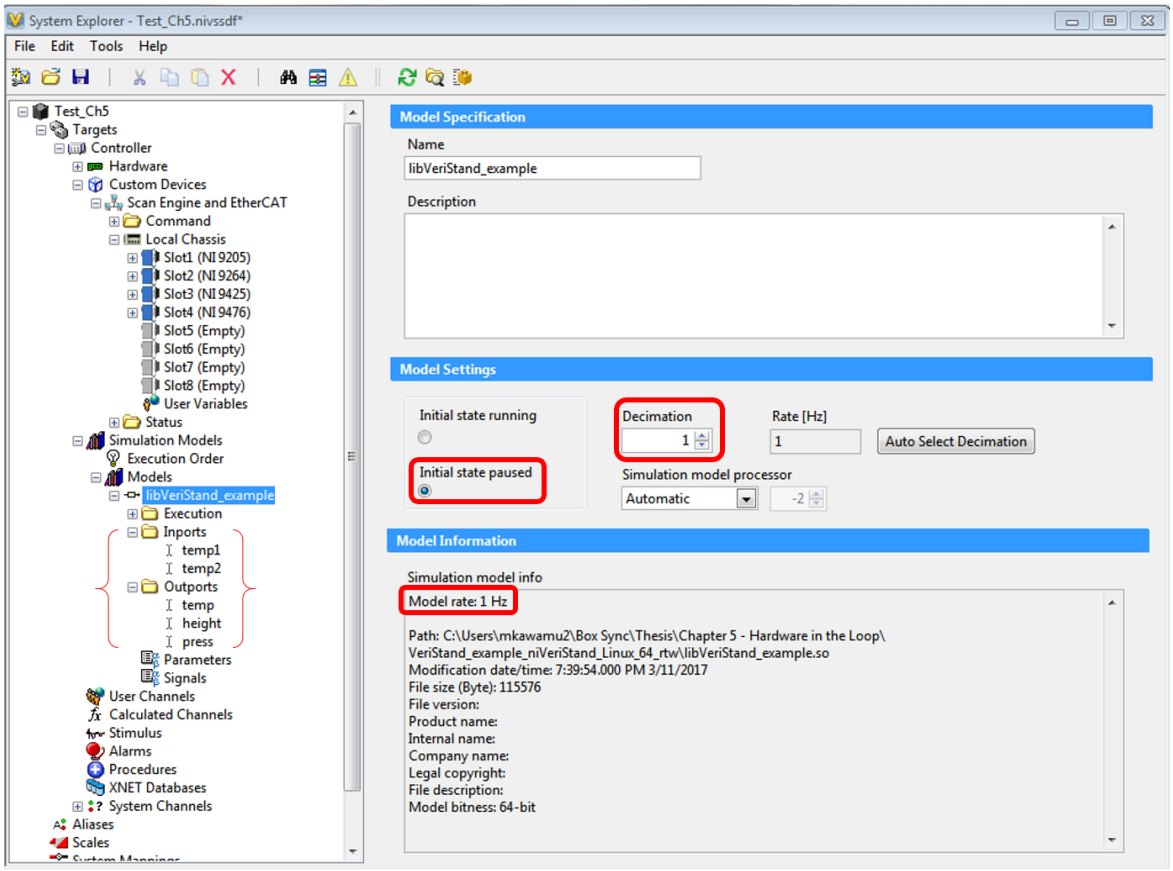

Figure 5.7 Example VeriStand System Definition file model specifications page after the simulation model is added.

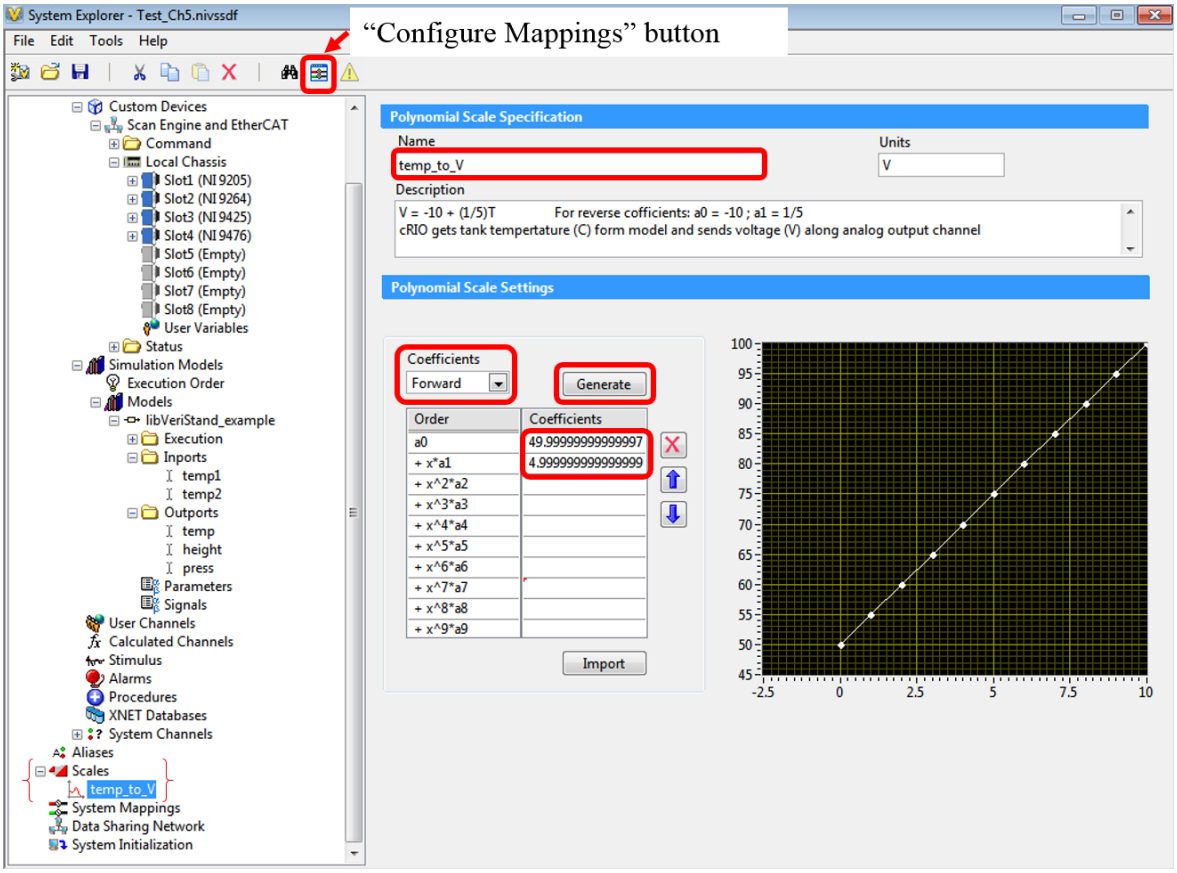

Figure 5.8 Example VeriStand System Definition file creating a custom scale example with notes about linear scaling and showing the how to generate the other coefficients.

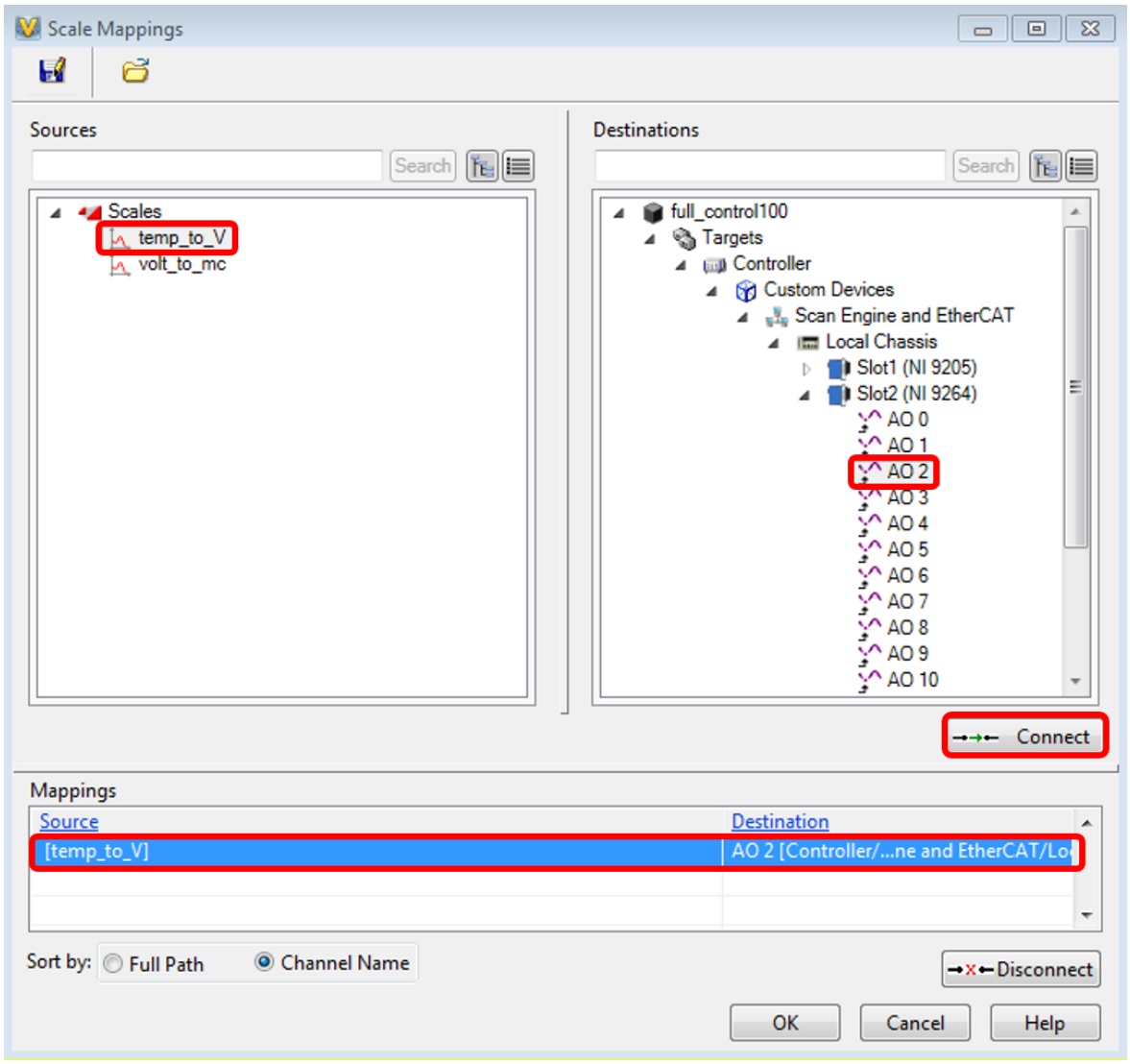

Figure 5.9 Example VeriStand System Definition file showing how to map a scale (temp_to_V) to a particular cRIO I/O channel (AO2).

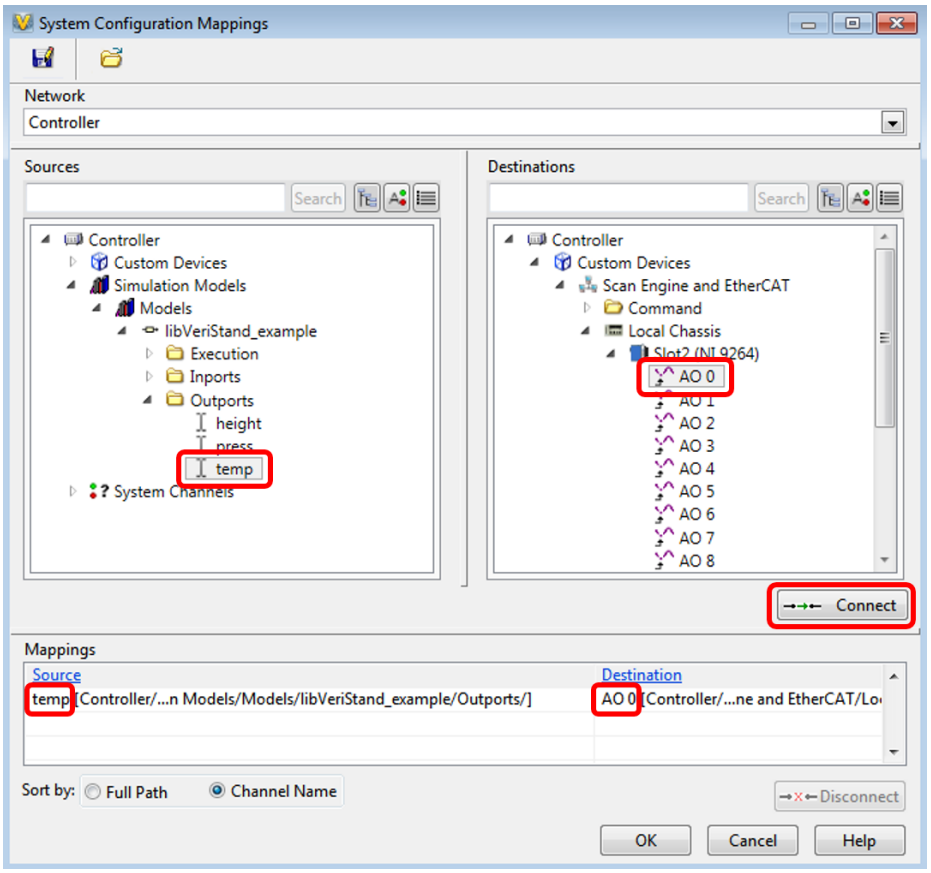

Figure 5.10 Example VeriStand System Definition file showing how to map model values to cRIO I/O channels.

Next, the User Interface Workspace file should be edited according to these steps. Note that the System Definition file must be closed before the User Interface file can be opened.

1. The UI file will not be used so one can right click on the *.nivsprj file to “Remove from Project”.
2. Open the User Interface Workspace file (*.nivsscreen).
3. To switch to edit mode press ctrl+m or Screen -> Edit Mode and a grey squared background will appear rather than a white background to denote that one is in the editing mode.
4. To add features, click on the “Workspace Controls” tab in upper left corner. It can be useful to pin this to the workspace by clicking the pin. Drag the desired objects onto Workspace by clicking on the names. An example Workspace in Edit Mode is shown in Figure 5.11, with components that are likely to be added.

5. Add the model control (Model -> Model Control). After dragging the control onto Workspace, map with the desired model already imported with the System Definition file.
6. Add the logging control (Logging -> Logging Control). Select the desired log file type in File Settings (CSV is what this project has used). Select the desired values to store in Channels. One can set a custom logging rate in Rate. It is often useful to set Start Trigger and Stop Trigger to be for a when condition is true (eg start when model time is greater than zero and stop when model time is greater than a set value) as shown in Figure 5.12.
7. Add any desired plots (Graph -> Simple). In the General tab, choose desired channels and set axes properties in the Format & Precision tab.
8. Add any desired analog controls (Numeric Control -> Medium). After dragging medium onto the workspace, map it to the desired channel.
9. When done editing the workspace, switch out of Edit Mode (ctrl+m) and save file.

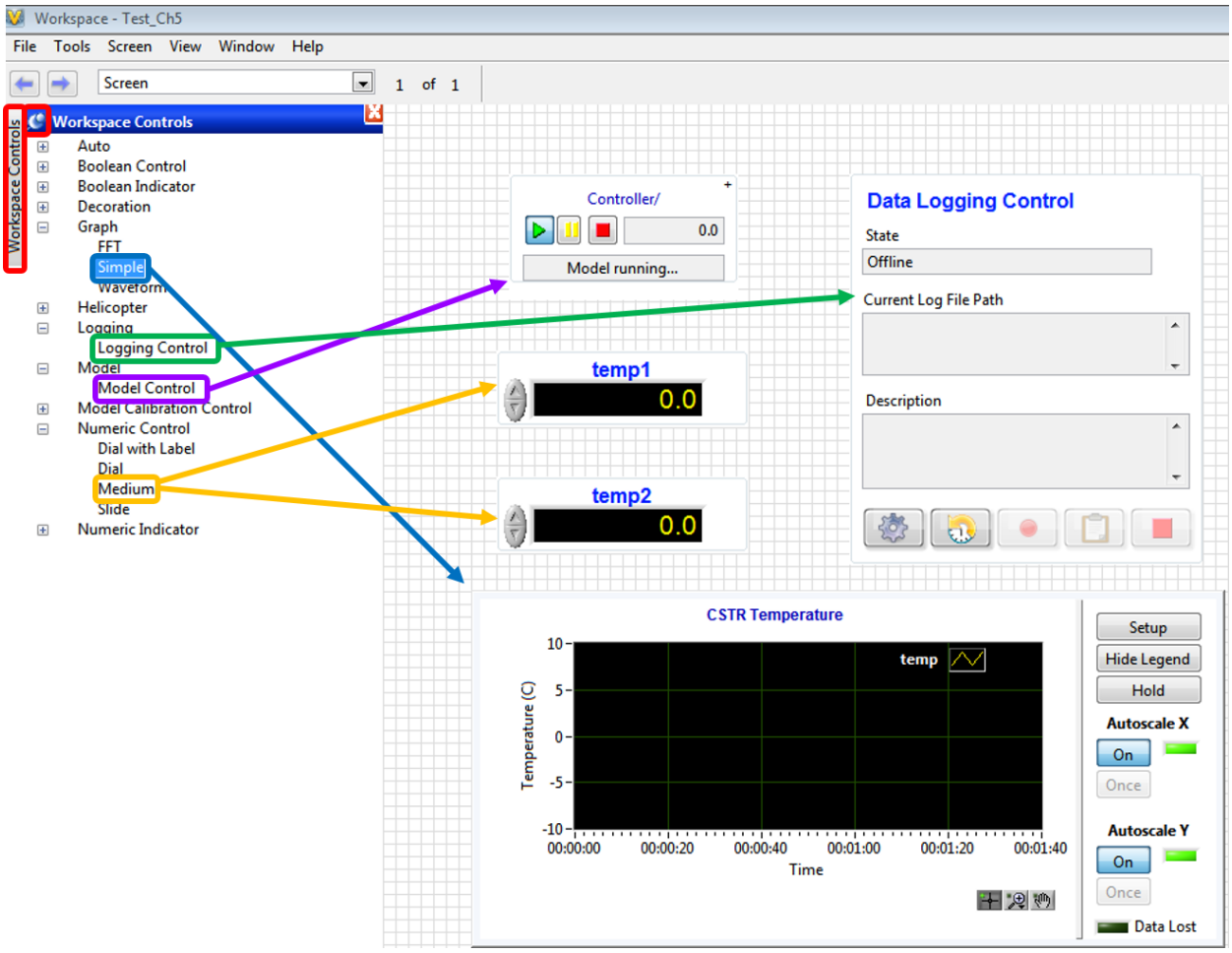

Figure 5.11 Example VeriStand Workspace showing components that are likely to be added.

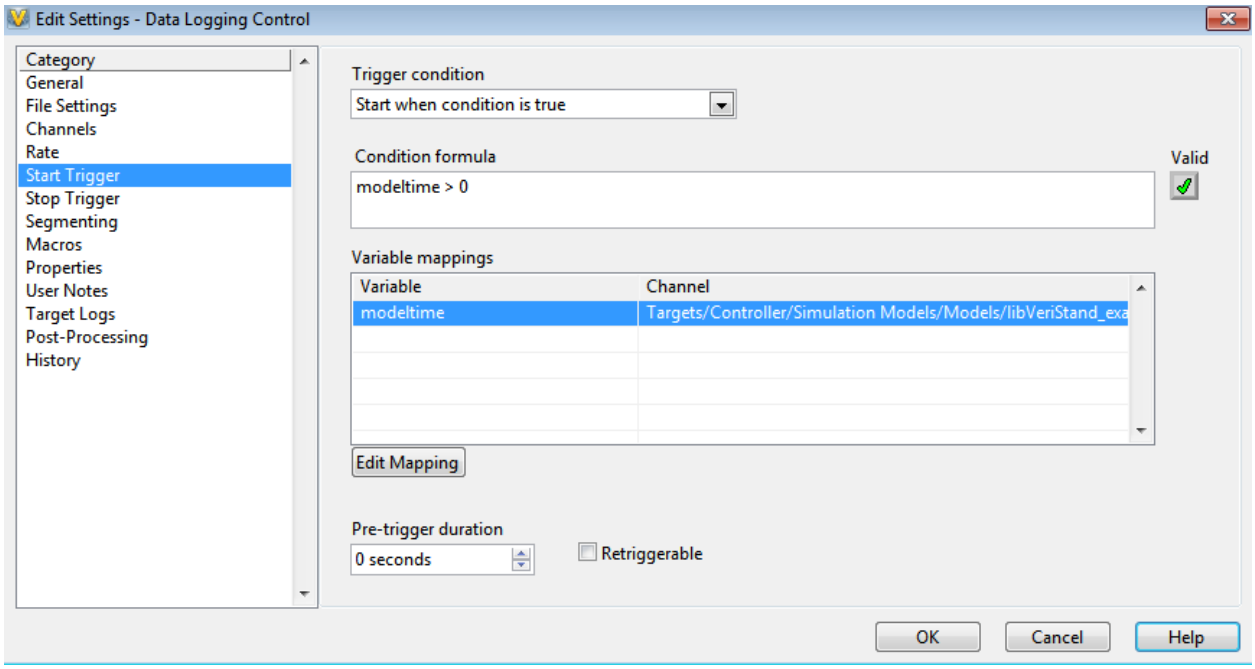

Figure 5.12 Example VeriStand Workspace Data Logging Control Start Trigger set to automatically start logging data when the model starts running.

With the System Definition and User Interface Workspace file complete, the model is ready to be run in real-time on the cRIO. To run the model, first deploy the System Definition file on the cRIO by using the Deploy button on the Project Explorer window. Deploying the System Definition file may take about a minute. After the deployment completes, start the data logging by pressing the red record button and then press the green play button on the model control to start the model. A useful debugging tool to ensure that the model is running at the desired rate is to include a plot of Model Count (found within System Channels) and if this plot is not zero, then it means the model is running late.

5.2.2 Controller

For this research, the controller did not need to support high computational loads. Therefore, a National Instruments myRIO was selected. The myRIO has built in analog inputs, analog outputs, digital inputs, and digital outputs and is fairly inexpensive at $250 since it is meant to be a student embedded device. The programming for the myRIO control is implemented in LabVIEW. A simple example VI which receives the tank temperature from the cRIO is shown in Figure 5.13. This example is consistent with the previous examples in Chapter

5. For the final HIL implementation, control would be implemented in the LabVIEW VI. However, for this example, the model inputs were handled using the VeriStand user workspace instead.

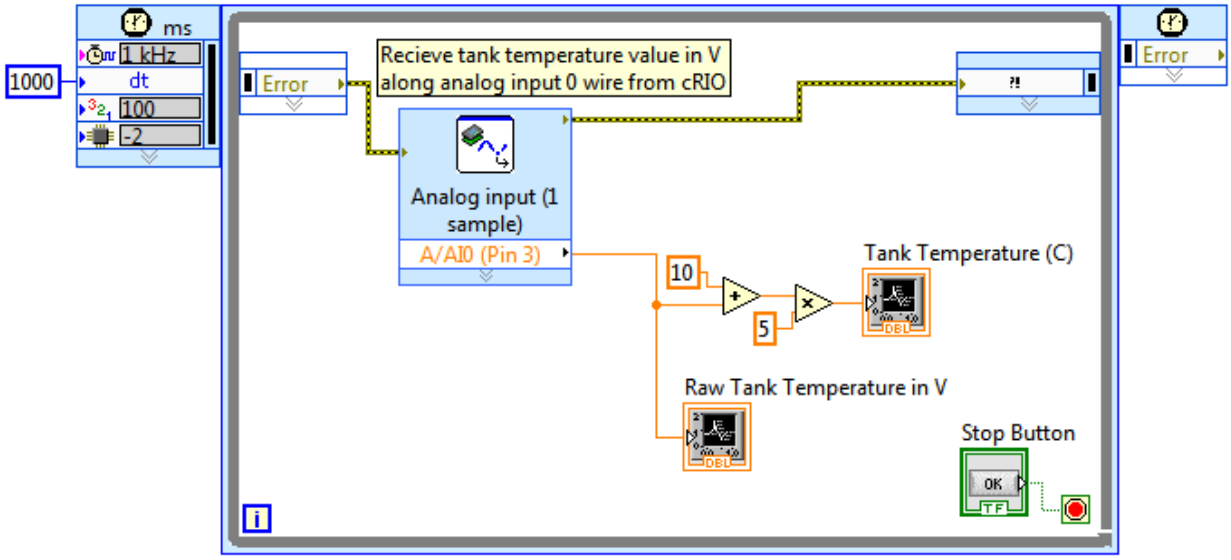

Figure 5.13 LabVIEW example VI for myRIO control showing how an analog input value can be received and converted from a voltage to a temperature in degrees Celsius.

The other primary hardware considered for the controller was a Siemens PLC. A PLC was considered because PLCs are an industry standard for the control of manufacturing processes [24], [29]. However, due to cost and not needing the additional computational power that PLCs provide, a myRIO was determined to be sufficient.

5.3 Hardware-in-the-Loop Formulation

Now it has been established that a cRIO will act as the embedded real-time plant and a myRIO will serve as the controller. The connection between the cRIO and myRIO is physical wires which can include both analog and digital signals. Figure 5.14 summarizes how this project utilizes the different hardware and software to run a chemical plant model in real-time and test different control algorithms. Additionally, Figure 5.15 shows the real cRIO, myRIO, and wire connections with the green breakout boards used to access the cRIO I/O modules.

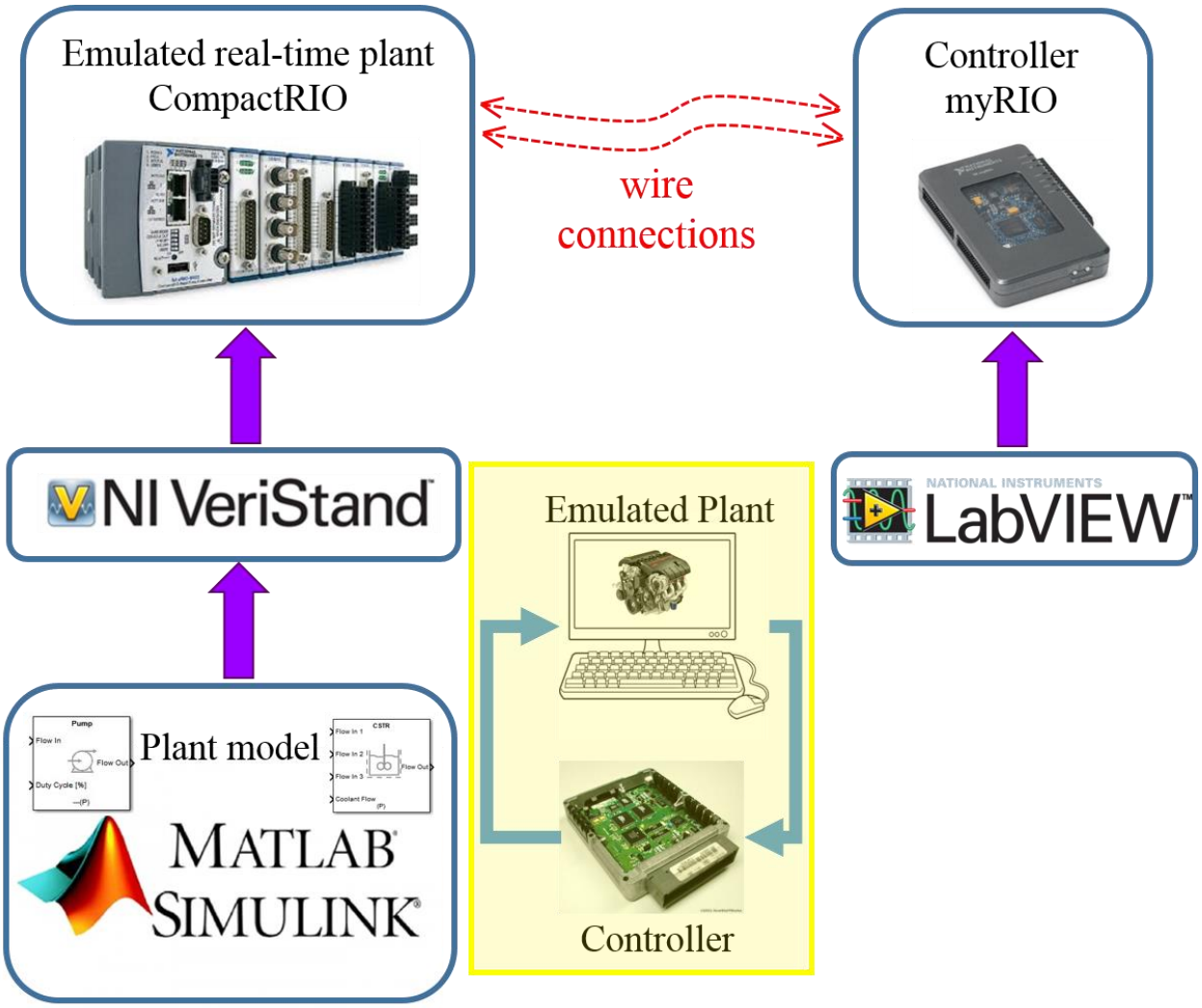

Figure 5.14 Schematic of HIL implementation discussed in Chapter 5 related back to the HIL implementation introduced in Chapter 1.

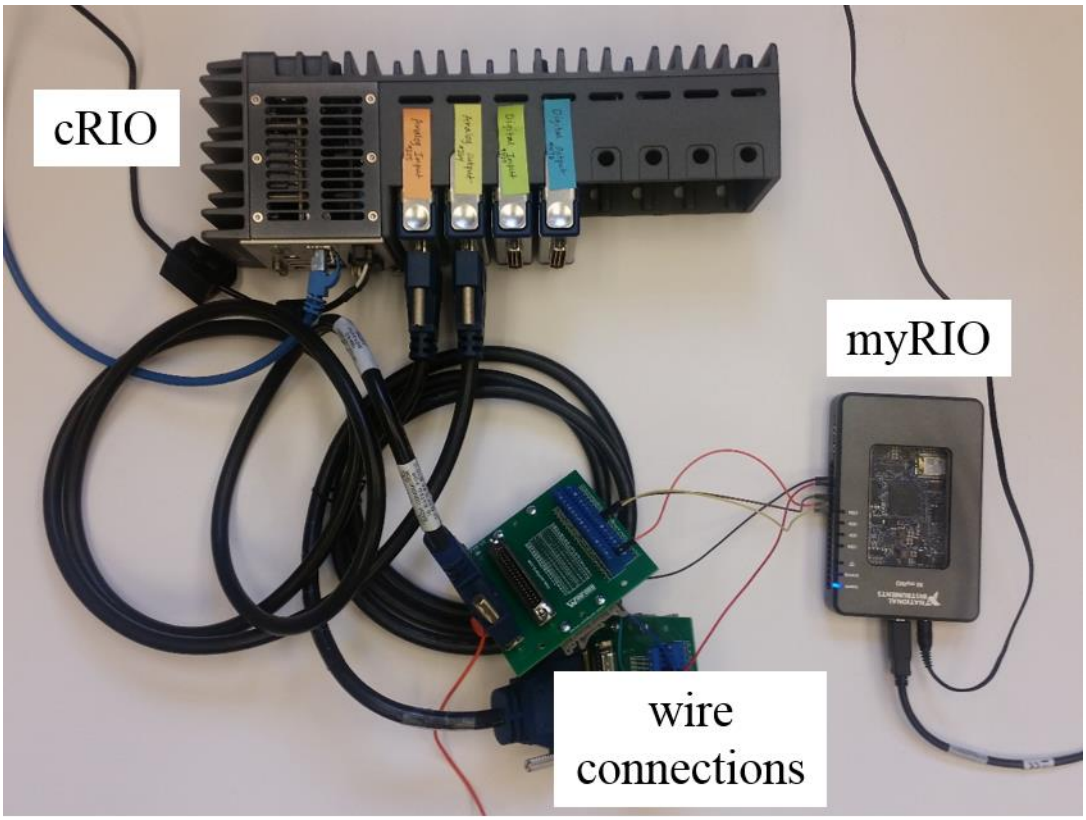

Figure 5.15 Physical HIL setup showing the cRIO with four I/O modules and myRIO with wires connecting the emulated real-time plant (cRIO) to the controller (myRIO).

Chapter 6
Implemented Control Strategy

With the process control plant example from Chapter 4 and the HIL implementation from Chapter 5, this chapter explains the applied control strategy, shares the HIL control results, and discusses future control opportunities. The framework, rather than the specific results, is the takeaway of this chapter. The results demonstrate how the HIL implementation performs as expected and control can be applied with the myRIO to the plant model running in real-time on the cRIO.

6.1 Plant Model and Control Strategy

The propylene glycol plant model from Chapter 4 shown in Figure 4.3 is run in real-time on the cRIO.

6.1.1 Control Input to Model

The chemical dynamics of the propylene glycol reaction depend heavily on the reaction temperature. The reaction temperature is strongly influenced by the heat transfer from the coolant fluid in the CSTR jacket to the contents inside the reactor. Recall that the heat transfer from the jacket exchanger to the reactor is given by

$$\dot{Q} = \dot{m}_c c_{pc}\{(T_{a1} - T)[1 - \exp(\frac{-UA_r}{\dot{m}_c c_{pc}})]\}, \tag{2.27}$$

where $\dot{m}_c$ is the coolant mass flow rate, c_{pc} is the coolant heat capacity, T_{a1} is the coolant jacket inlet temperature, T is the temperature in the reactor, U is the jacket heat transfer coefficient,

and A_r is the heat transfer area between the jacket and reactor as shown in Figure 6.1. Note that in Figure 6.1, if there are multiple inlet flows, all inlet flows are simplified to be at an average temperature, T_0, and at a combined flow rate F_{A0}. However, if this is not the desired case, it is simple to modify the system to have heterogeneous inlet flows. For large mass flow rates, the exponential term in Equation (2.27) is small and using a Taylor series expansion and neglecting second-order terms, the simplified equation for heat transfer from the jacket to the reactor given by

$$\dot{Q} = UA_r(T_a - T), \tag{2.4}$$

where $T_a = T_{a1} \approx T_{a2}$. The simplified equation, (2.4), is provided because this assumption is often used in textbooks, but this model does implement the heat transfer equation in (2.27). To influence this heat transfer, the coolant mass flow rate ($\dot{m}_c$) is treated here as the control input to the plant.

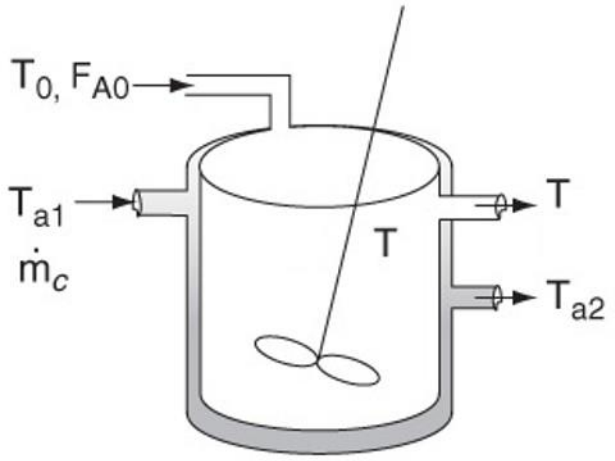

Figure 6.1 A schematic showing how the CSTR jacket interacts with the CSTR [37]**.**

6.1.2 Control Implementation

The coolant mass flow rate as the control input means that the myRIO controller sends an analog signal to the cRIO, the emulated real-time plant as shown in Figure 6.2. To calculate the controller output, the coolant mass flow rate, the myRIO needs to know the temperature in the reactor. Therefore, the cRIO plant model must send an analog voltage that represents the

temperature to the myRIO. These signals are each sent every one second. The cRIO and myRIO can both send and receive analog voltages between -10V and +10V. The temperature and coolant mass flow rate analog signals sent are linearly scaled to be within the -10V to +10V range.

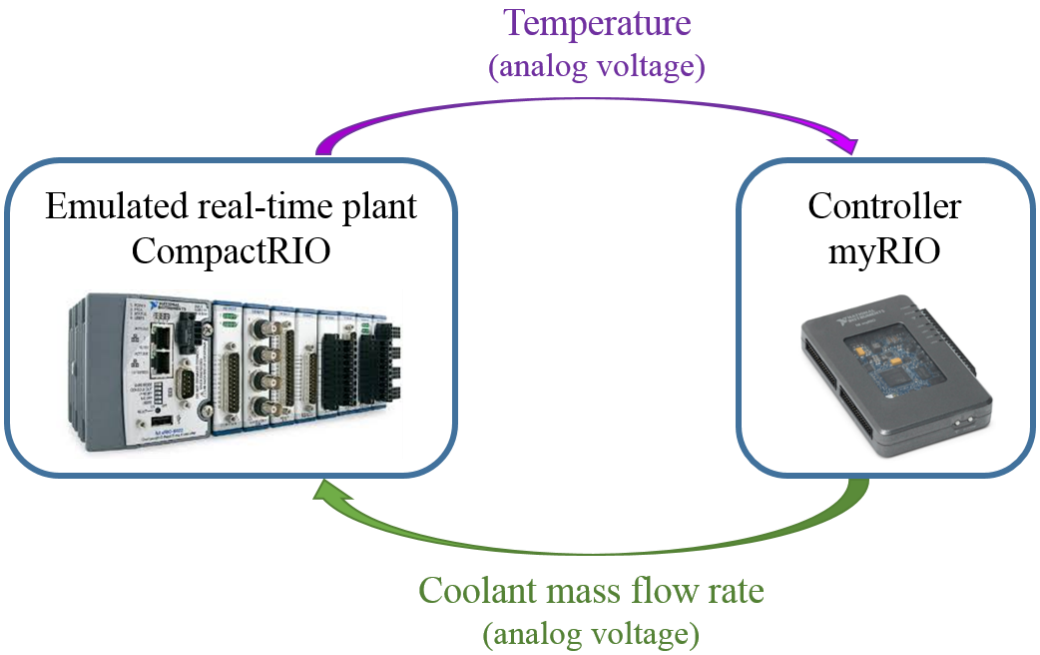

Figure 6.2 A schematic of the control implementation with a myRIO controller and a cRIO emulated propylene glycol real-time plant.

6.1.3 Control Algorithm

To determine the coolant mass flow rate control input, the myRIO acts as a PI controller. The input signal to the PI controller is the error signal,

$$T_{ref} - T, \tag{6.1}$$

where $T_{ref} = 58.8°C$ is the reference temperature and T is the reactor temperature. In LabVIEW, the PI control is implemented as shown in Figure 6.3 with the same anti-windup based on back-calculation used in the Simulink simulation previously. The plant has a negative gain, since as the coolant mass flow rate increases the temperature decreases. Therefore, negative proportional and integral gains are expected. Specifically, the gains set on the LabVIEW front

panel were a proportional gain of $P = -30$, integral gain of $I = -0.008$, and back calculation coefficient of $kb = 0.008$.

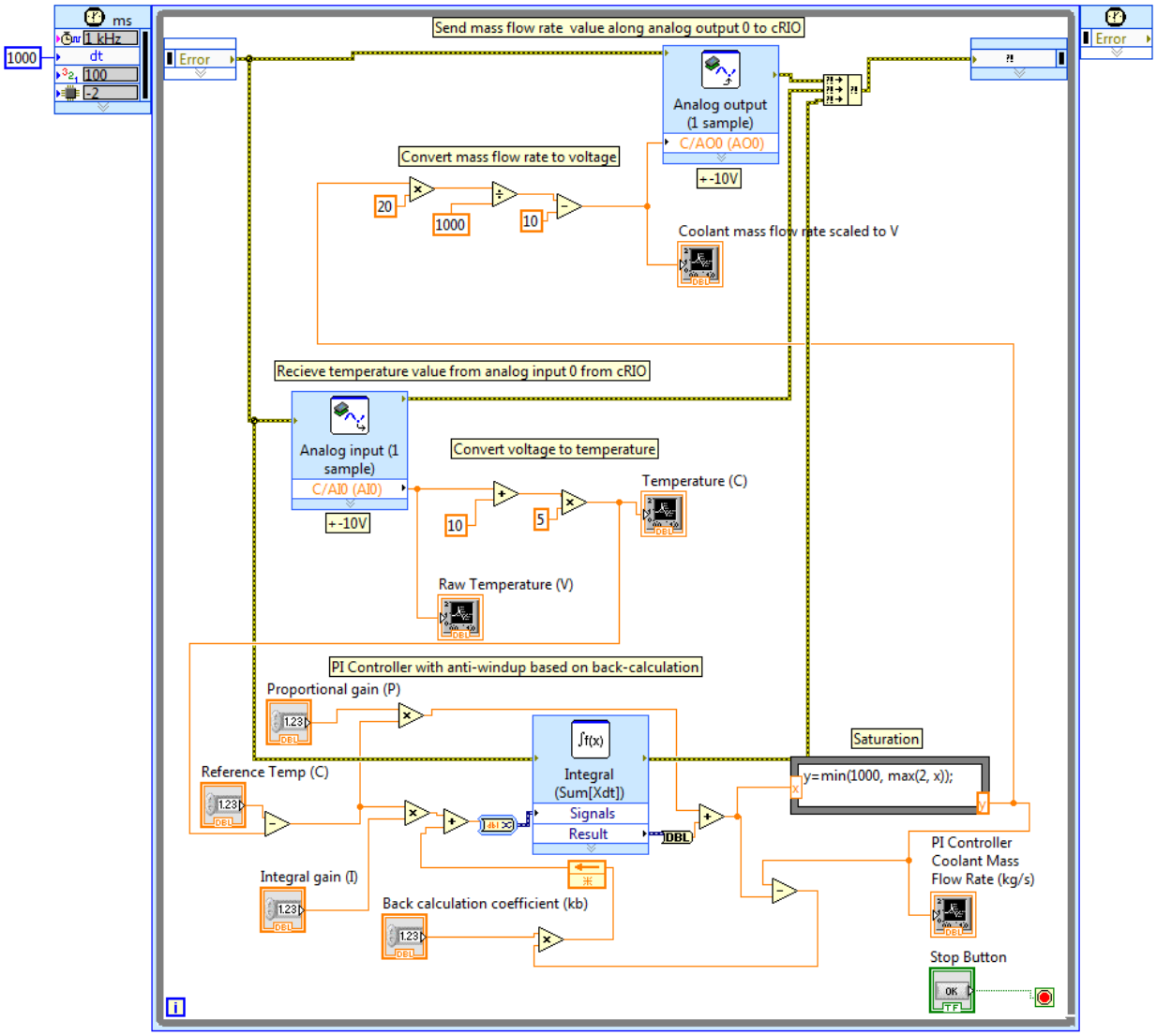

Figure 6.3 LabVIEW code for PI control with anti-windup based on back-calculation which is run on a myRIO.

6.2 HIL Control Results

With the HIL configuration shown in Figure 6.2, the following results were obtained. Figures 6.5-6.9 each show how the plant model responds when the coolant mass flow rate is controlled or set at a constant value of 1000 kg/s as shown in Figure 6.4. The PI controller

demonstrated was empirically tuned by the author and is not necessarily the optimal choice of gains. That said, the results show how a lower mass flow rate can be obtained with a controller to regulate to the desired reactor temperature of $58.8°C$. Additionally, with the controller, the steady state temperature and concentration values are reached sooner, which could be useful.

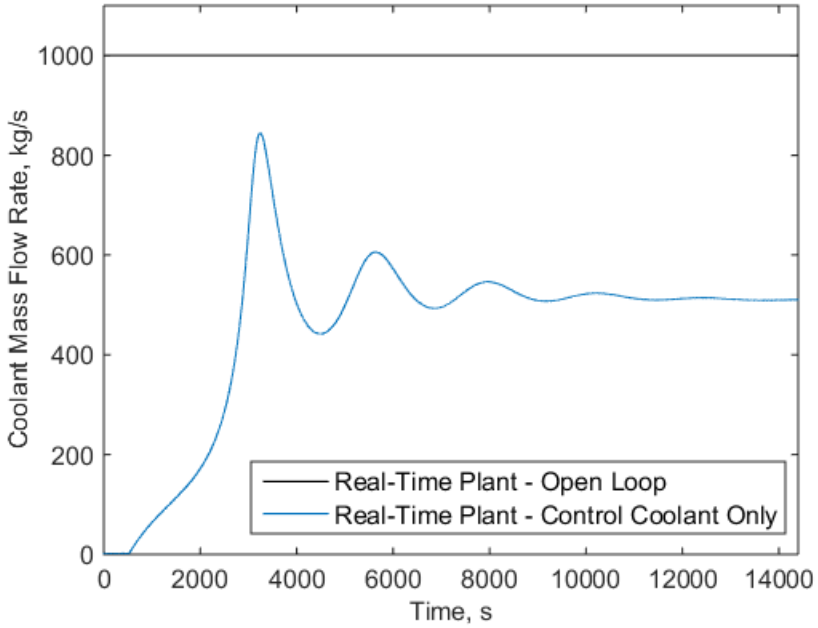

Figure 6.4 Coolant mass flow rate versus time for propylene glycol reaction with PI coolant mass flow rate control and run in open-loop.

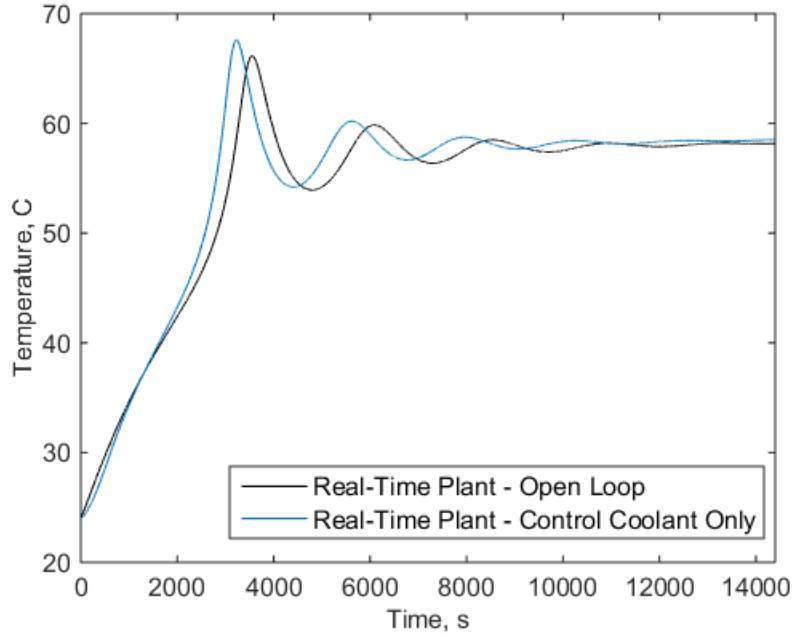

Figure 6.5 Reaction temperature versus time for propylene glycol reaction with PI coolant mass flow rate control and run in open-loop.

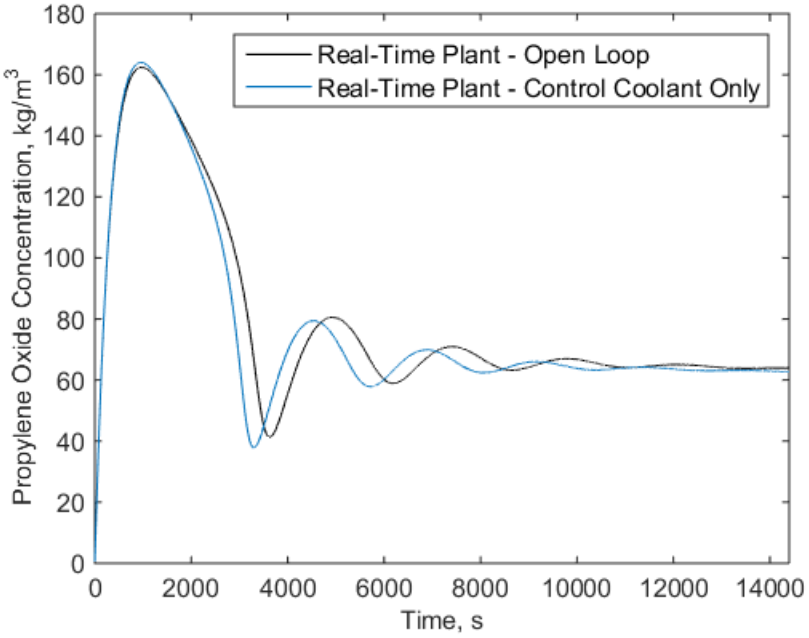

Figure 6.6 Propylene oxide concentration versus time for propylene glycol reaction with PI coolant mass flow rate control and run in open-loop.

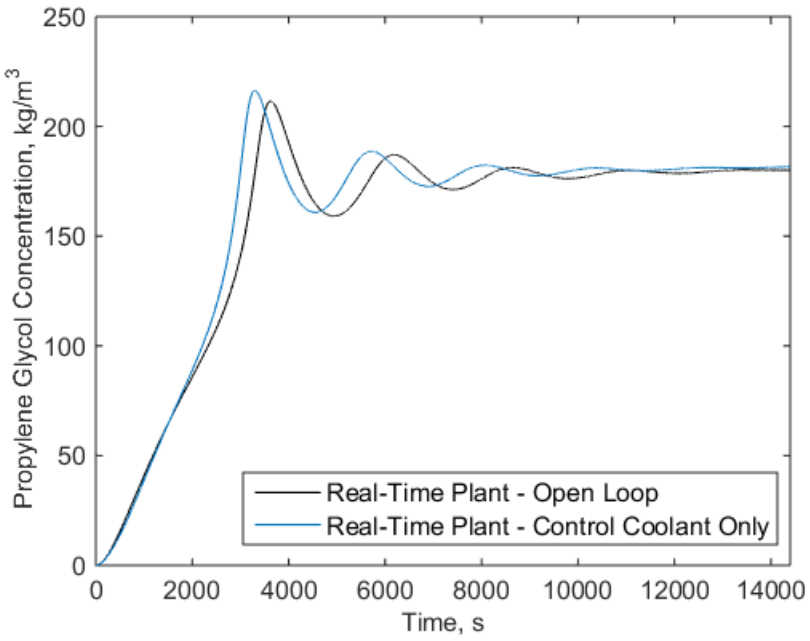

Figure 6.7 Propylene glycol concentration versus time for propylene glycol reaction with PI coolant mass flow rate control and run in open-loop.

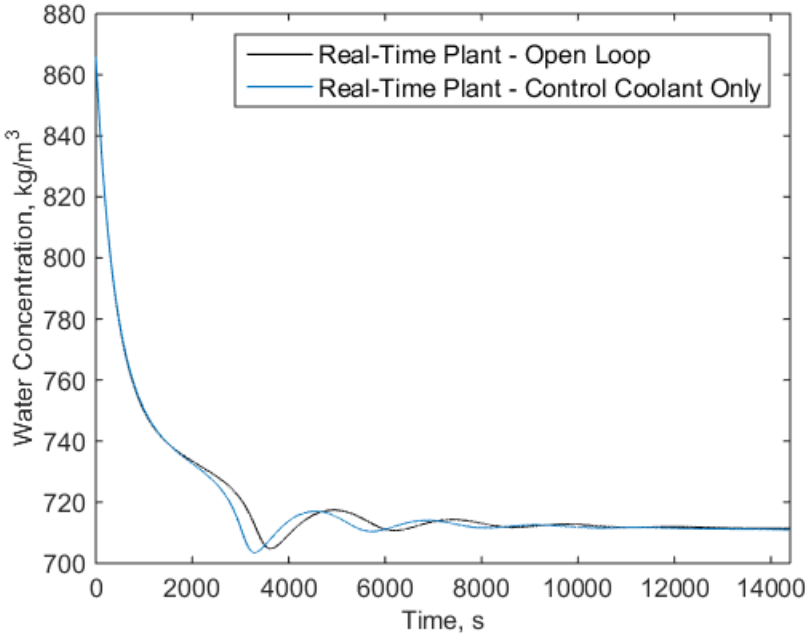

Figure 6.8 Water concentration versus time for propylene glycol reaction with PI coolant mass flow rate control and run in open-loop.

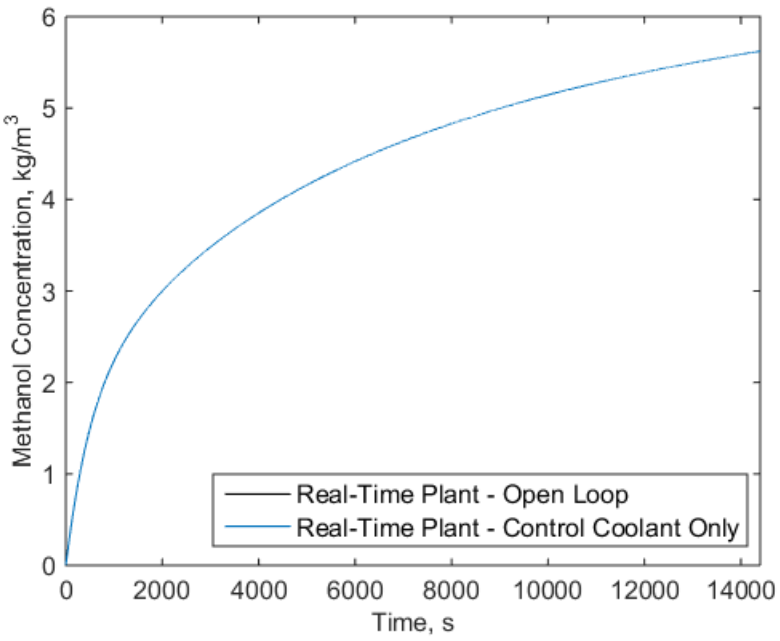

Figure 6.9 Methanol concentration versus time for propylene glycol reaction with PI coolant mass flow rate control and run in open-loop.

6.3 HIL Generalization – Parameter Variation

To demonstrate that the results in Section 6.2 work for different models, this section shows additional results for varying the heat transfer coefficient, U, between the jacket and reactor tank. The heat transfer coefficient is set at 100% the original value, 110% the original value, and 120% the original value. The controller gains vary for the different heat transfer coefficients as shown in Table 6.5. However, the limits on the controller output between 2 kg/s and 1000 kg/s remained constant for all trials. Figures 6.10-6.15 show that the framework is generalizable for different models.

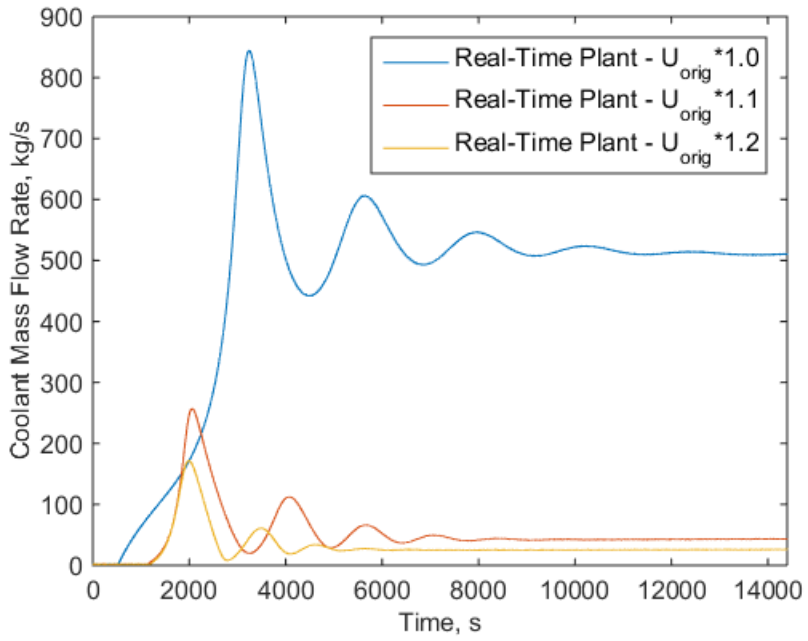

Figure 6.10 Coolant mass flow rate versus time for propylene glycol reaction with PI control for the real-time plant for varying heat transfer coefficient values.

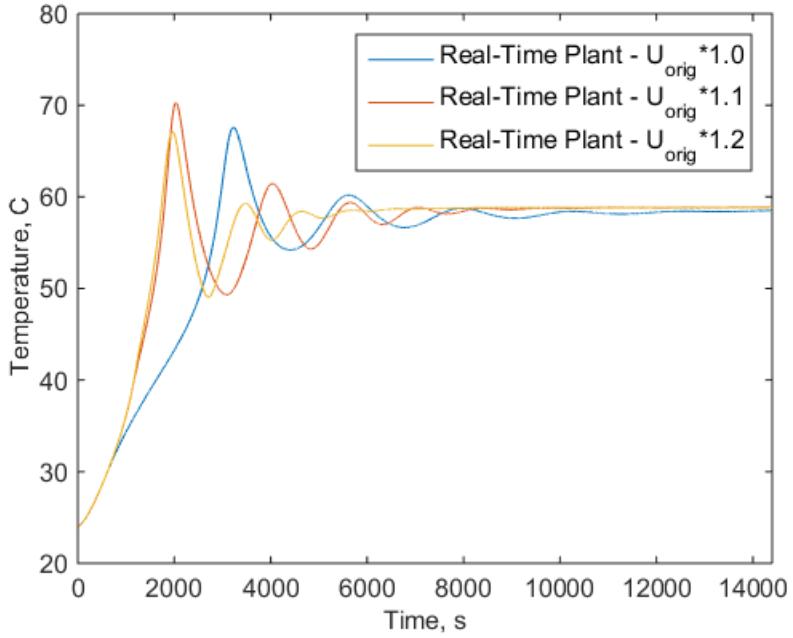

Figure 6.11 Reaction temperature versus time for propylene glycol reaction with PI control for the real-time plant for varying heat transfer coefficient values.

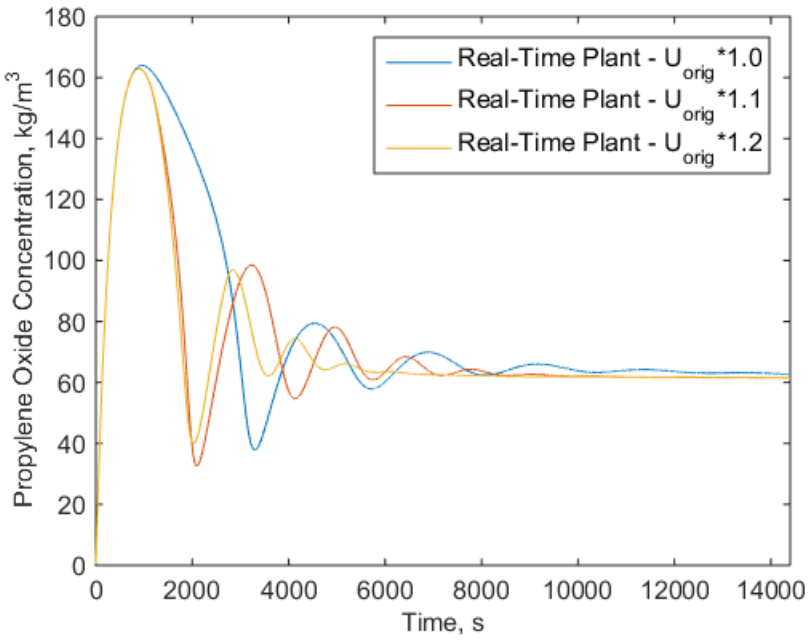

Figure 6.12 Propylene oxide concentration versus time for propylene glycol reaction with PI control for the real-time plant for varying heat transfer coefficient values.

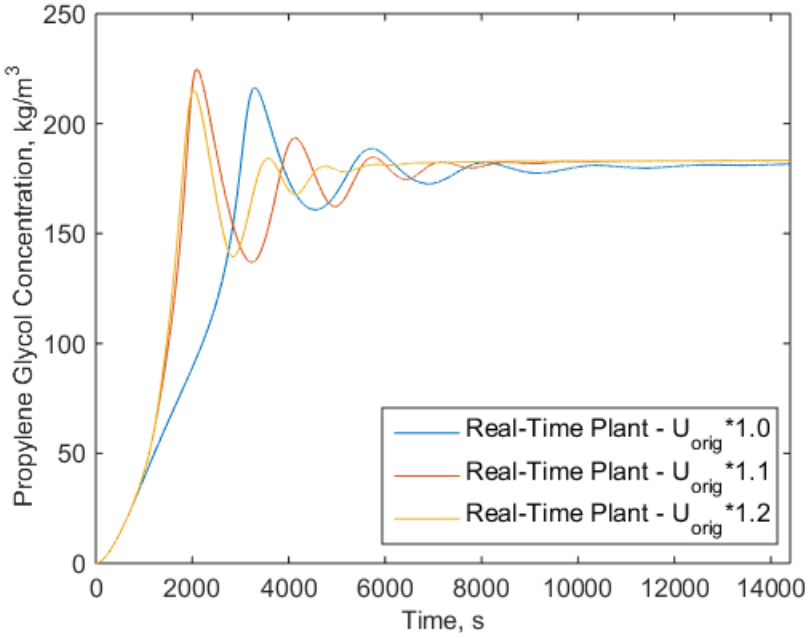

Figure 6.13 Propylene glycol concentration versus time for propylene glycol reaction with PI control for the real-time plant for varying heat transfer coefficient values.

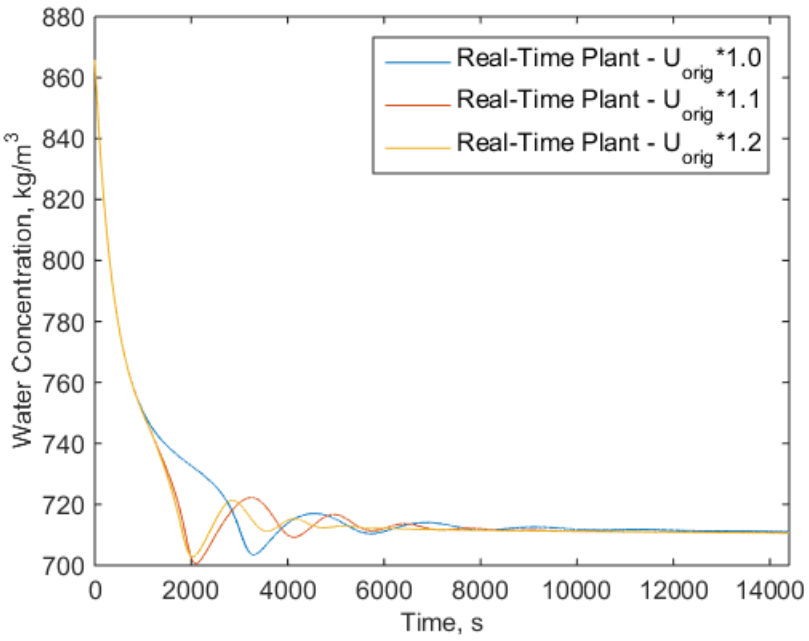

Figure 6.14 Water concentration versus time for propylene glycol reaction with PI control for the real-time plant for varying heat transfer coefficient values.

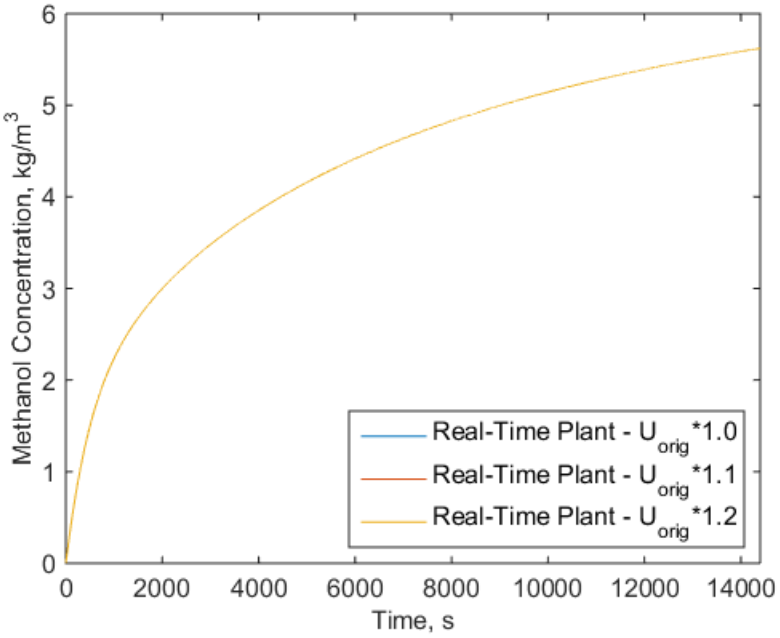

Figure 6.15 Methanol concentration versus time for propylene glycol reaction with PI control for the real-time plant for varying heat transfer coefficient values.

Table 6.5 Controller settings for the HIL testing and simulation parameter variation of the heat transfer coefficient where U_{orig} is the original heat transfer coefficient value.

Trial	Proportional Gain	Integral Gain	Kb	Saturation Limits
U_{orig} *1.0	-30	-0.008	0.008	2-1000
U_{orig} *1.1	-10	-0.01	0.01	2-1000
U_{orig} *1.2	-8	-0.01	0.01	2-1000

6.4 HIL and Simulation Result Comparison

As expected, the results are identical for the same control applied both in simulation and run in real-time on hardware. The LabVIEW code shown in Figure 6.3 shows how a PI controller also with anti-windup based on back-calculation was applied on the real-time plant since anti-windup based on back-calculation was used for the simulated plant results in Simulink. Figures 6.16-6.21 show the good agreement between the HIL and pure simulation results for varying heat transfer coefficient values.

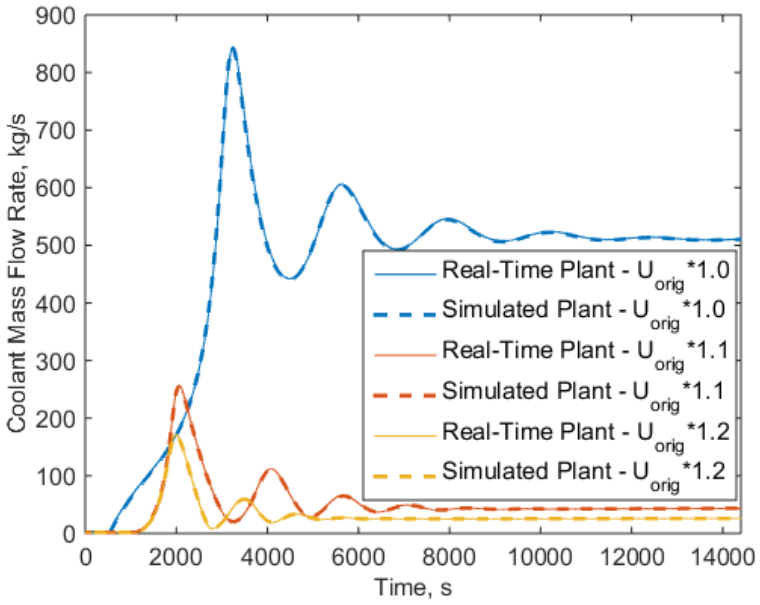

Figure 6.16 Coolant mass flow rate versus time for propylene glycol reaction with PI control for the real-time plant on a cRIO and simulated plant on a computer for varying heat transfer coefficient values.

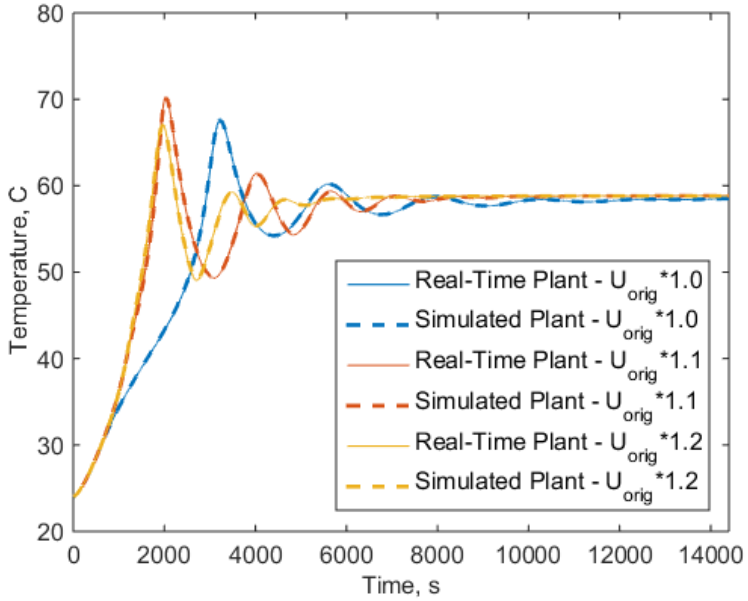

Figure 6.17 Reaction temperature versus time for propylene glycol reaction with PI control for the real-time plant on the cRIO and simulated plant on a computer for varying heat transfer coefficient values.

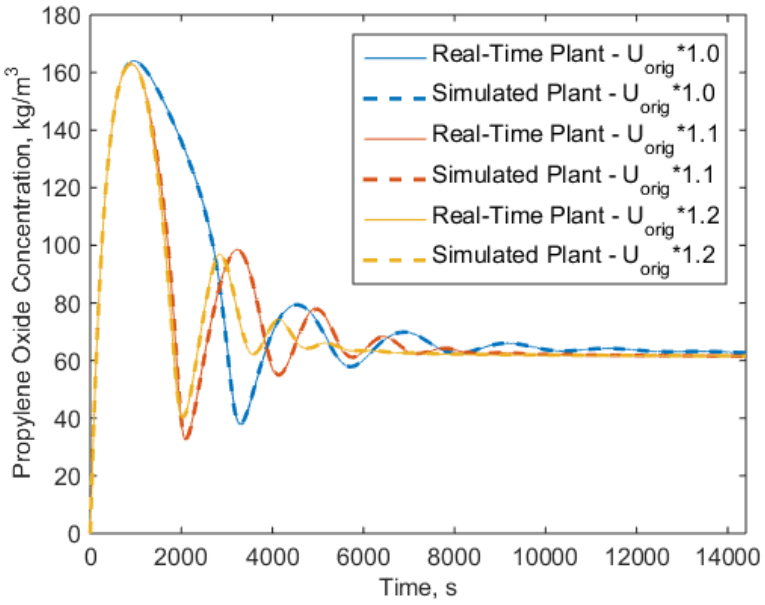

Figure 6.18 Propylene oxide concentration versus time for propylene glycol reaction with PI control for the real-time plant on a cRIO and simulated plant on a computer for varying heat transfer coefficient values.

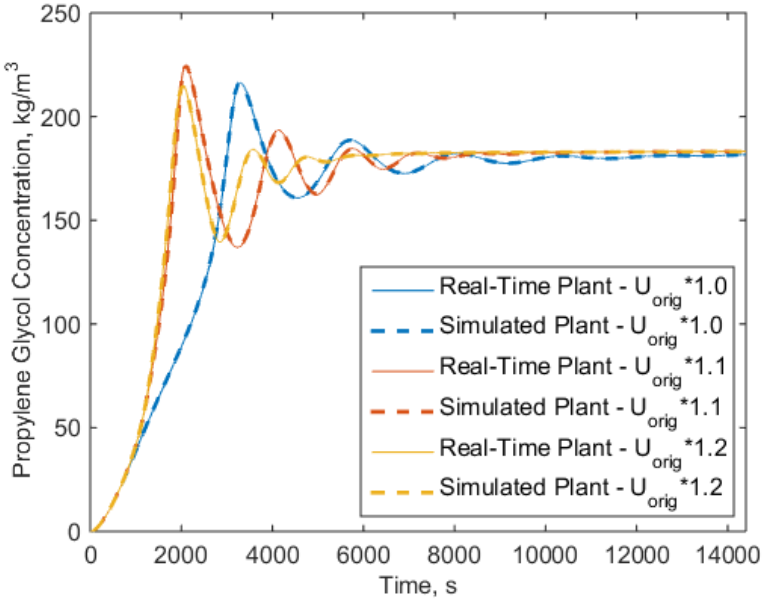

Figure 6.19 Propylene glycol concentration versus time for propylene glycol reaction with PI control for the real-time plant on a cRIO and simulated plant on a computer for varying heat transfer coefficient values.

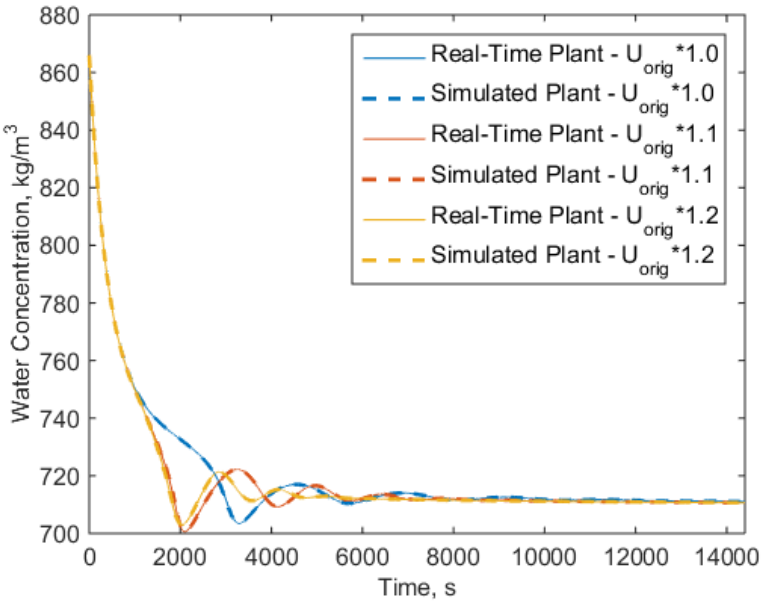

Figure 6.20 Water concentration versus time for propylene glycol reaction with PI control for the real-time plant on a cRIO and simulated plant on a computer for varying heat transfer coefficient values.

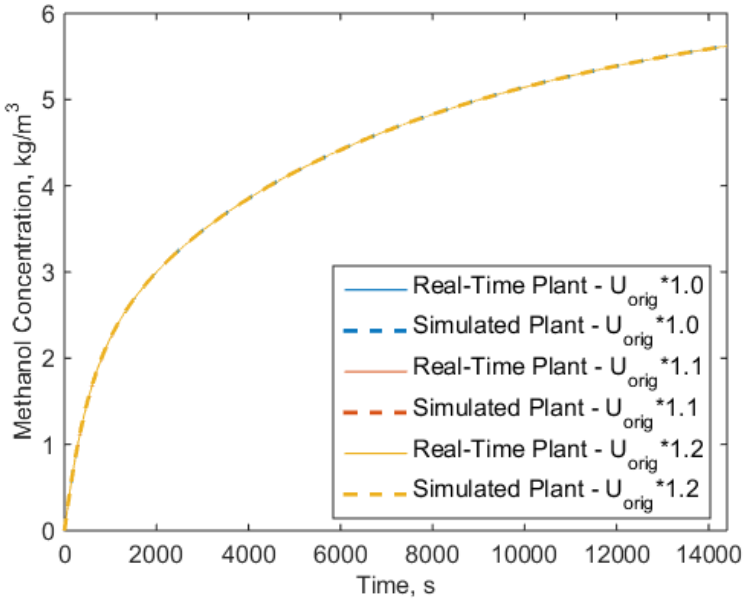

Figure 6.21 Methanol concentration versus time for propylene glycol reaction with PI control for the real-time plant on a cRIO and simulated plant on a computer for varying heat transfer coefficient values.

6.5 Future Control Opportunities

This HIL configuration is set up to accommodate a variety of other control opportunities in the future. The hardware is capable of having many additional signals beyond the single analog output from the myRIO and single analog output from the cRIO. At this point, the myRIO would be the limiting factor with six analog output channels, ten analog inputs channels, and forty digital I/O channels total.

Specifically, a cascaded control to regulate the reactor temperature could be a valuable addition. Figure 6.22 shows an example of how cascaded control could be applied to regulate the reactor temperature perhaps more effectively than with a single PI controller as implemented already. The cascaded control architecture requires an "early warning" process variable which in this example case is the jacket outlet temperature. An advantage of this architecture should be improved disturbance rejection [51].

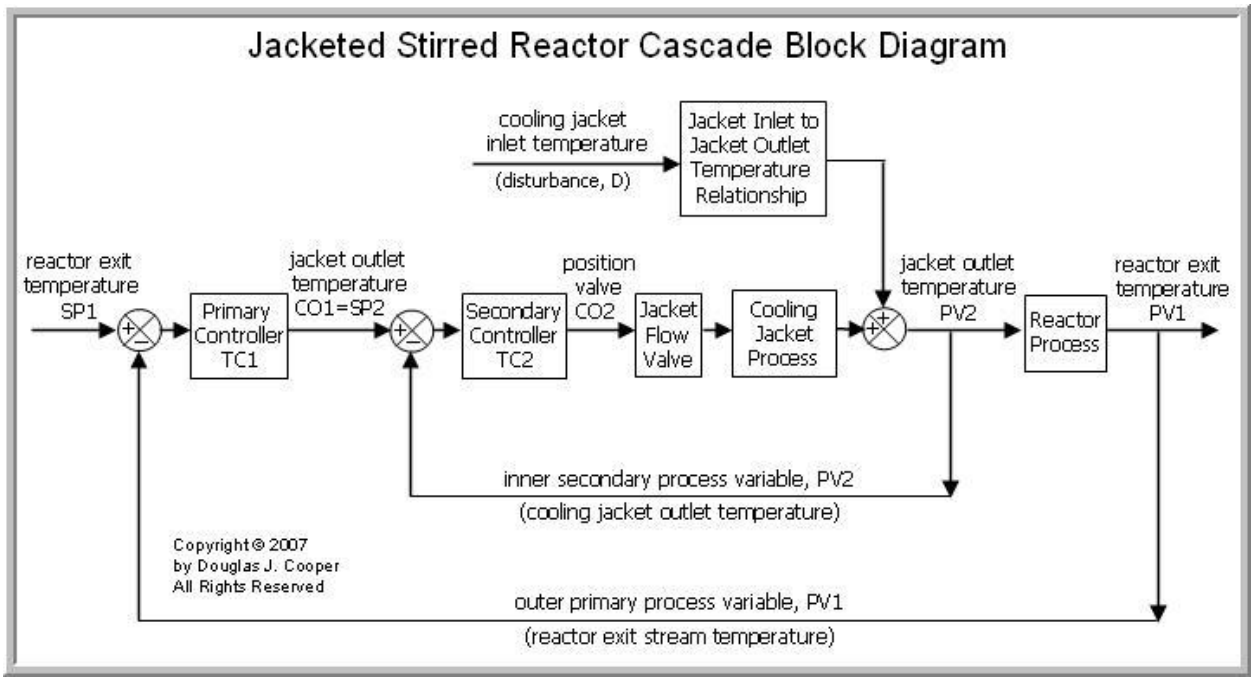

Figure 6.22 Block diagram example of a cascaded control algorithm for a reactor with a jacket to regulate the reactor temperature [51]**.**

Chapter 7
Conclusion

7.1 Summary of Research Contributions

This thesis develops a HIL testing framework for chemical processes. Advantages of the HIL framework include faster development time, cost savings, increased safety, and bridging the gap between simulation and experimental systems. Creating this framework involved two main thrusts – the dynamic modeling of a chemical plant and the hardware-in-the-loop implementation and control.

7.1.1 Dynamic Modeling Toolkit

To have a modular, extendable, and scalable modeling toolkit, a library of commonly found chemical plant components was created in Matlab Simulink. The nonlinear components include a continuous stirred tank reactor (CSTR), CSTR jacket, pump, valve, pipes, mass source and sink, and pressure source and sink. These initial components allow testing of a variety of plant configurations and chemical reactions to be investigated quickly. A key strength of this modeling toolkit is that the parameters for the user to enter have real physical meaning. Other educational toolkits often lack physical meaning and do not provide as much flexibility in modeling complex chemical reactions [34], [35]. Additionally, this toolkit is computationally efficient enough to be run in real-time, which is necessary for the HIL testing. Validation of the concentration in a reactor was done experimentally using the colorimetric crystal violet reaction and some other components were previously validated. This dynamic modeling toolkit provides first principles models that can run in real-time.

7.1.2 HIL Implementation and Control

To implement HIL testing, hardware was selected for the emulated real-time plant and controller. A cRIO was selected to be the emulated real-time chemical plant. A main challenge was how to run a Simulink model on the cRIO in real-time. To bridge the gap between the Matlab Simulink model and National Instruments cRIO hardware, NI VeriStand was utilized. VeriStand converted the Simulink model to run in real-time on the cRIO and provided a user interface to connect cRIO physical input and output I/O channels to different model inputs and outputs.

For the controller, a NI myRIO was utilized because the applied PI control required low computational power. The control for the myRIO was developed in LabVIEW. Analog signals were communicated between the cRIO (emulated real-time plant) and myRIO (controller) via physical wires. The wired connection completed the HIL implementation.

This implementation provided a proof of concept for HIL testing, with no emphasis on finding the "best" control for the example plant model. Additionally, this thesis provided a very detailed description of how to make the transfer from a Simulink model running on a desktop computer to a model running in real-time on a cRIO.

7.2 Future Work

Future work should include modeling toolkit development and validation as well as additional control algorithm development.

7.2.1 Modeling and Validation

It could be very useful to develop additional reactors beyond a CSTR to model additional reaction processes. Additionally, a main assumption in the developed toolkit is that no chemical reactions happen outside of the reactor. However, the reactions are likely to continue in the pipes and other components after the reactor and this dynamic behavior could be important to include for particular plant structures. As with other simulation tools, such as Thermosys^{TM}, effective results could be made without altering the pipe blocks [52]. For example, a single pipe block could be 'cut' into two pipe blocks with a fictitious reactor placed in the middle. Then the pipes would be responsible for the pressure drop and mass flow with the reaction happening in the

fictitious reactor. This would be a temporary fix to allow current blocks to emulate reactions continuing outside the reactor. A longer term effort would involve modeling pipes with reactions.

Validation of an entire plant process could be verified with experimental data in addition to the component by component validation discussed in Chapter 3. Additionally, validation for the temperature of an exothermic or endothermic reaction could be valuable in addition to the crystal violet reaction validation for the concentration in a CSTR.

7.2.2 Control

Developing control methods was not a goal of this work. However, with the modeling toolkit developed and HIL testing implementation complete, there is an opportunity for more complex control methods to be rapidly tested. The example of cascaded control provided in Chapter 6 is one possible future control direction with SISO control. Additional possibilities include, but are not limited to, multi-input multi-output (MIMO) control and MPC. For chemical reactions, precise mass flow rates into and out of a CSTR are very important which means that MIMO control could be especially useful for controlling the overall mass flow rates.

References

[1] M. Ellis, H. Durand, and P. D. Christofides, "A tutorial review of economic model predictive control methods," *J. Process Control*, vol. 24, no. 8, pp. 1156–1178, 2014.

[2] United States Environmental Protection Agency, "Clean Power Plan," 2017. [Online]. Available: https://www.epa.gov/cleanpowerplan.

[3] U.S. Energy Information Administration, "Manufacturing Energy Consumption Survey," 2002. [Online]. Available: https://www.eia.gov/consumption/manufacturing/briefs/chemical/.

[4] R. Isermann, J. Schaffnit, and S. Sinsel, "Hardware-in-the-loop simulation for the design and testing of engine-control systems," *Control Eng. Pract.*, vol. 7, no. 5, pp. 643–653, 1999.

[5] J. Kocijan, G. Žunič, S. Strmčnik, and D. Vrančić, "Fuzzy gain-scheduling control of a gas-liquid separation plant implemented on a PLC," *Int. J. Control*, vol. 75, no. 14, pp. 1082–1091, 2002.

[6] A. White, G. Zhu, and J. Choi, "Hardware-in-the-loop simulation of robust gain-scheduling control of port-fuel-injection processes," *IEEE Trans. Control Syst. Technol.*, vol. 19, no. 6, pp. 1433–1443, 2011.

[7] J. Zhang, T. Li, A. Amodio, B. Aksun-Guvenc, and G. Rizzoni, "Hardware-in-the-Loop Implementation of Fault Diagnosis and Fault Tolerant Control Strategies for Electrified Vehicle Torque Functional Safety," ASME Dynamic Systems and Control Division Newsletter, Nov-2016.

[8] P. Terwiesch, T. Keller, and E. Scheiben, "Rail Vehicle Control System Integration Testing Using Digital Hardware-in-the-Loop Simulation," *IEEE Trans. Control Syst. Technol.*, vol. 7, no. 3, pp. 352–362, 1999.

[9] Z. Li, M. Kyte, and B. Johnson, “Hardware-in-the-loop real-time simulation interface software design,” *Proceedings. 7th Int. IEEE Conf. Intell. Transp. Syst. (IEEE Cat. No.04TH8749)*, pp. 1012–1017, 2004.

[10] M. V. Americano da Costa, M. Pasamontes, J. E. Normey-Rico, J. L. Guzmán, and M. Berenguel, “Advanced Control Strategy Combined with Solar Cooling for Improving Ethanol Production in Fermentation Units,” *Ind. Eng. Chem. Res.*, vol. 53, no. 28, pp. 11384–11392, 2014.

[11] T. Zang and A. Wang, “Design and application of the Hardware-in-the-Loop Simulation System for Distillation Columns,” *2009 Int. Conf. Image Anal. Signal Process.*, pp. 372–376, 2009.

[12] D. M. Lima, M. V. Americano da Costa, and J. E. Normey-Rico, “A Flexible Low Cost Embedded System for Model Predictive Control of Industrial Processes,” *Eur. Control Conf.*, pp. 1571–1576, 2013.

[13] M. V. Americano da Costa, M. Pasamontes, J. E. Normey-Rico, J. L. Guzmán, and M. Berenguel, “Viability and application of ethanol production coupled with solar cooling,” *Appl. Energy*, vol. 102, pp. 501–509, 2013.

[14] W. Bequette, *Process Control: Modeling, Design, and Simulation*. Upper Saddle River: Prentice Hall Professional, 2003.

[15] J. L. Fernandes, C. I. . C. Pinheiro, N. M. C. Oliveira, J. Inverno, and F. R. Ribeiro, “Closed-loop Dynamic Behavior of an Industrial High-Efficiency FCC Unit,” *2008 Mediterr. Conf. Control Autom. - Conf. Proceedings, MED’08*, pp. 558–563, 2008.

[16] O. Levenspiel, *Chemical Reaction Engineering*, Third Edit. John Wiley & Sons, Inc., 1999.

[17] University of York, “Chemical Reactors,” 2013. [Online]. Available: http://www.essentialchemicalindustry.org/processes/chemical-reactors.html.

[18] H. S. Fogler, *Elements of Chemical Reaction Engineering*, Second Edi. Prentice Hall PTR, 1992.

[19] T. E. Marlin, *Process Control - Designing Processes and Control Systems for Dynamic Performance*, First Edit. McGraw-Hill, Inc., 1995.

[20] Wikipedia, “Lime kiln,” 2017. [Online]. Available:

https://en.wikipedia.org/wiki/Lime_kiln.

[21] Aspentech, “Aspen HYSYS,” 2016. [Online]. Available: http://www.aspentech.com/products/aspen-hysys/.

[22] M. V. Americano da Costa and T. L. M. Santos, “Using Solar Irradiation for Steam Generation in Bioethanol Production : an Initial Study,” in *2015 6th International Renewable Energy Congress (IREC)*, 2015, pp. 1–6.

[23] H. Khodadadi and H. Jazayeri-Rad, “Applying a dual extended Kalman filter for the nonlinear state and parameter estimations of a continuous stirred tank reactor,” *Comput. Chem. Eng.*, vol. 35, no. 11, pp. 2426–2436, 2011.

[24] W. Grega, “Hardware-in-the-loop simulation and its application in control education,” in *FIE’99 Frontiers in Education. 29th Annual Frontiers in Education Conference. Designing the Future of Science and Engineering Education. Conference Proceedings (IEEE Cat. No.99CH37011*, 1999, vol. 2, p. 12B6/7-12B612.

[25] Z. Song, W. Jun, C. Liu, and X. Song, “Design of a Hardware-in the Loop Experiment Simulation System for Process Control based on RTW,” in *Proceedings of the 1st International Workshop on Education Technology and Computer Science, ETCS 2009*, 2009, pp. 1123–1126.

[26] A. F. Yazdi, “Modeling Industrial Chemical Processes with MATLAB and Simulink,” *MathWorks Technical Articles and Newsletters*, pp. 1–4, 2012.

[27] MathWorks, “Simulink,” 2016. [Online]. Available: https://www.mathworks.com/products/simulink.html.

[28] J. J. Downs and E. F. Vogel, “A Plant-Wide Industrial Process Control Problem,” *Comput. Chem. Eng.*, vol. 17, no. 3, pp. 245–255, 1993.

[29] G. Valencia-Palomo and J. A. Rossiter, “Programmable logic controller implementation of an auto-tuned predictive control based on minimal plant information,” *ISA Trans.*, vol. 50, no. 1, pp. 92–100, 2011.

[30] dSPACE GmbH, “dSPACE,” 2016. [Online]. Available: https://www.dspace.com/en/pub/home.cfm.

[31] N. Ginot, J. C. Le Claire, and L. Loron, “Active loads for Hardware in the Loop emulation of electro-technical bodies,” in *31st Annual Conference of IEEE Industrial Electronics*

Society (IECON), 2005.

[32] M. V. Americano da Costa and J. E. Normey-Rico, "Modeling, Control and Optimization of Ethanol Fermentation Process," in *Proceedings of the 18th IFAC World Congress*, 2011, vol. 44, no. 1, pp. 10609–10614.

[33] G. Valencia-Palomo and J. A. Rossiter, "Auto-tuned Predictive Control Based on Minimal Plant Information," in *IFAC Proceedings Volumes*, 2009, vol. 42, no. 11, pp. 554–559.

[34] R. K. Herz, "Reactor Lab," 2016. [Online]. Available: reactorlab.net.

[35] B. Postlethwaite, "The use of simulation in chemical process control learning and the development of PISim," in *2016 American Controls Conference (ACC)*, 2016, pp. 7328–7333.

[36] D. E. Seborg, T. A. Edgar, and D. A. Mellichamp, *Process Dynamics and Control*, First Edit. New York: John Wiley & Sons, Inc., 1989.

[37] H. S. Fogler, *Elements of Chemical Reaction Engineering*, Fifth Edit. 2016.

[38] J. P. Koeln, M. A. Williams, H. C. Pangborn, and A. G. Alleyne, "Experimental Validation of Graph-Based Modeling For Thermal Fluid Power Flow Systems," in *Proceedings of the ASME 2016 Dynamic Systems and Control Conference*, 2016, pp. 1–10.

[39] Valvias, "Flow Coefficient Definition," 2013. [Online]. Available: http://www.valvias.com/flow-coefficient.php.

[40] Spirax Sarco Limited, "Control Valve Characteristics," *2017*. [Online]. Available: http://www.spiraxsarco.com/Resources/Pages/Steam-Engineering-Tutorials/control-hardware-el-pn-actuation/control-valve-characteristics.aspx.

[41] Pasco, "Order of Reaction Laboratory Handout." pp. 1–18, 2016.

[42] Adafruit, "Peltier Thermo-Electric Cooler Module + Heatsink Assembly," 2017. [Online]. Available: https://www.adafruit.com/product/1335.

[43] A. E. Martin and F. H. Murphy, "Glycols : Propylene Glycols," 200AD.

[44] Merchant Research & Consulting ltd, "Propylene Glycol Production Will Be Dominated by the US Companies," 2015. [Online]. Available: https://mcgroup.co.uk/news/20150910/propylene-glycol-production-dominated-companies.html.

[45] Yskin, “PID Intro,” 2017. [Online]. Available: http://rtocare.tistory.com/entry/PID-Intro.

[46] H. S. Fogler, “Unsteady State Nonisothermal Reactor Design,” *4th Edition Elements of Chemical Reaction Engineering*, 2008. [Online]. Available: http://umich.edu/~elements/09chap/frames.htm.

[47] G. ten Broeke, G. van Voorn, and A. Ligtenberg, “Which Sensitivity Analysis Should I Use for My Agent-Based Model?,” *J. Artif. Soc. Soc. Simul.*, vol. 19, no. 1, 2016.

[48] MathWorks, “Anti-Windup Control Using a PID Controller,” 2017. [Online]. Available: https://www.mathworks.com/help/simulink/examples/anti-windup-control-using-a-pid-controller.html.

[49] National Instruments, “The CompactRIO Platform,” 2017. [Online]. Available: http://www.ni.com/compactrio/.

[50] National Instruments, “VeriStand,” 2017. [Online]. Available: http://www.ni.com/veristand/.

[51] C. Douglas, “Practical Process Control,” 2015. [Online]. Available: http://controlguru.com/a-cascade-control-architecture-for-the-jacketed-stirred-reactor/.

[52] B. P. Rasmussen, “Dynamic Modeling and Advanced Control of Air Conditioning and Refrigeration Systems,” University of Illinois at Urbana-Champaign, 2005.

Appendix A
Simulink Library Code

The Simulink library code has the structure shown in Figure A.1. Detailed descriptions of the user interactions with each component's mask is provided in Section 2.3. The code for each component is included here. Each component has a mask. With the mask, each component can have initialization code. In each mask, parameters can also be included and some parameters have callbacks associated with them. The code in the callbacks allow the user interface to automatically update immediately when the user makes changes, unlike the initialization code. Finally, beneath each mask is the subsystem, which has a unique structure for each component. The structure is displayed through images and the code for any Matlab functions is included.

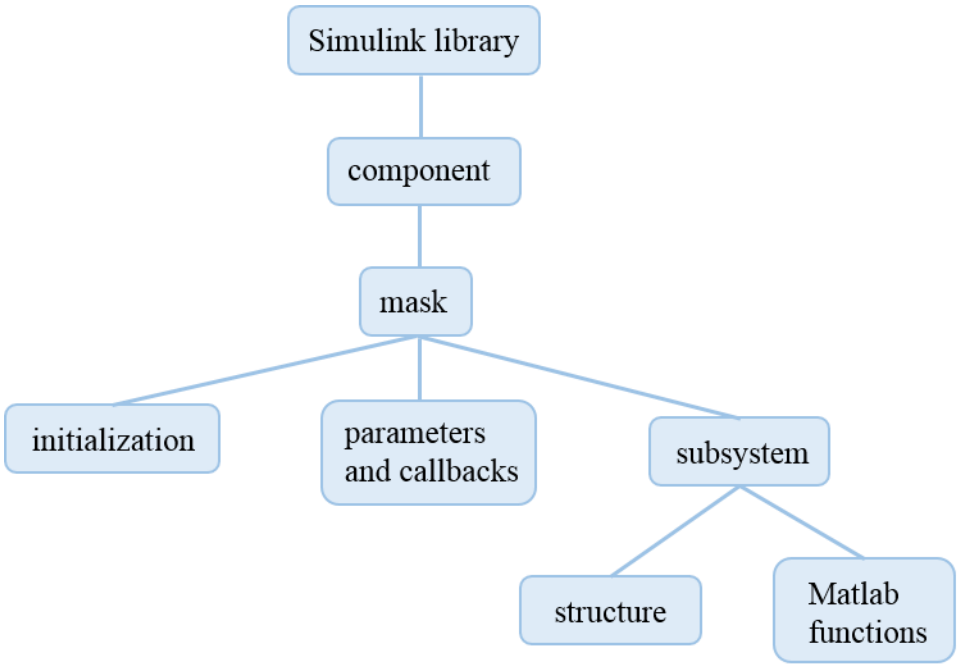

Figure A.1 Structure of the Simulink library code showing how each component in the library can have an initialization, parameters and callbacks, and a subsystem.

A.1 Continuous Stirred Tank Reactor

A.1.1 Initialization

```
% load fluid properties
load('Fluids.mat');

% Calculate geometric values
V = (pi()*d^2*hinit)/4;                 % liquid volume
set_param(gcb,'V',num2str(V));
Vt = (pi()*d^2*ht)/4;                   % tank volume
set_param(gcb,'Vt',num2str(Vt));
At = (2*pi()*(d/2)*ht);                 % surface area bewteen tank and heater
set_param(gcb,'At',num2str(At));
```

A.1.2 Parameters and Callbacks

Figures A.2 and A.3 show the CSTR mask parameters.

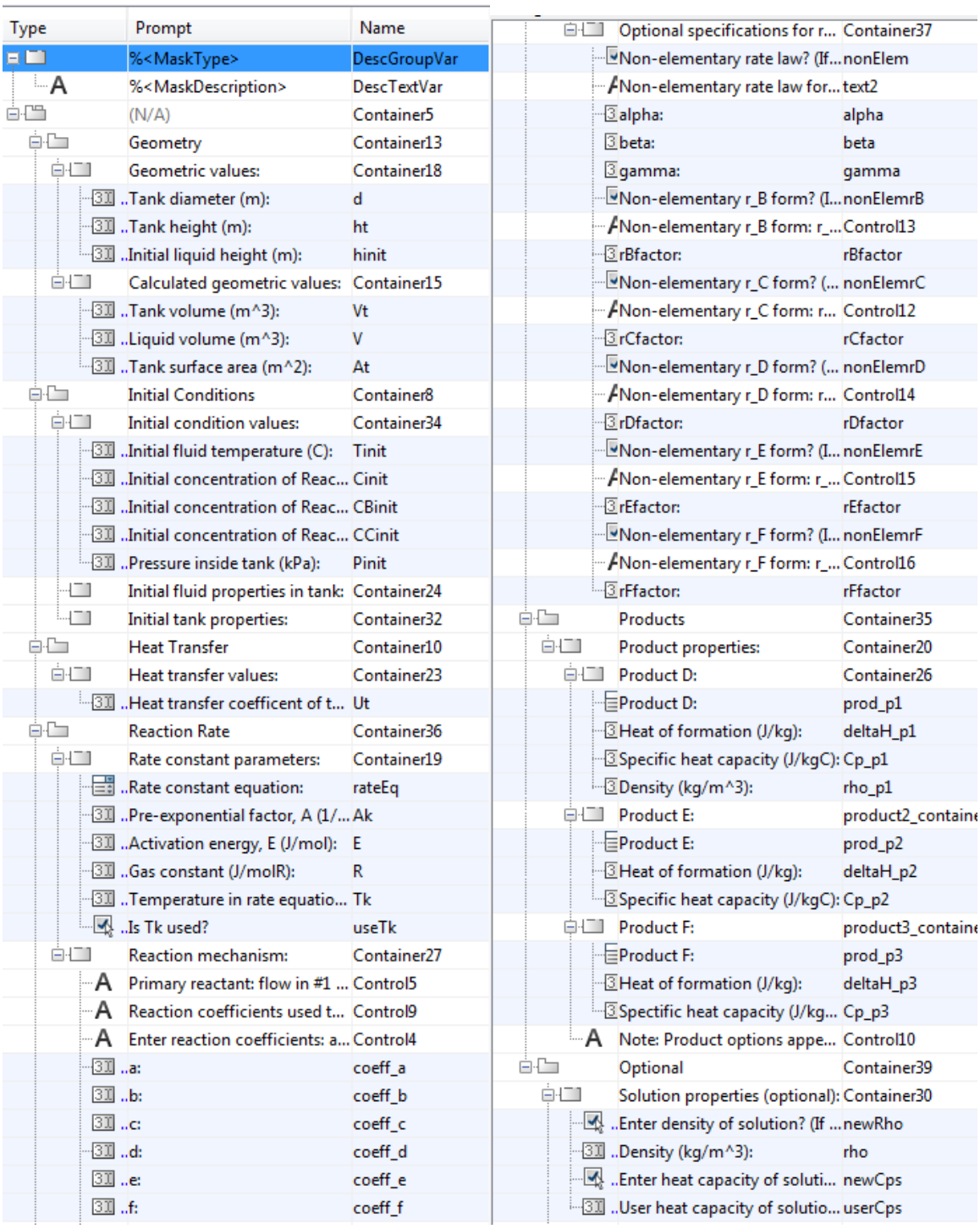

Figure A.2 CSTR mask parameters (1/2).

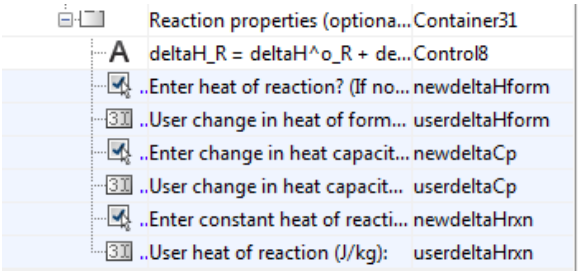

Figure A.3 CSTR mask parameters (2/2).

The CSTR has the following callbacks associated with the parameters.

rateEq callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'Tk'))
        index_coeff_Tk = i;
    elseif (strcmp(names(i),'useTk'))
        index_coeff_useTk = i;
    end
end

% Based on rate law selected by user, show Tk parameter to fill in or don't
% show Tk parameter. Set or don't set 'useTk' parameter to be used in
% function (invisible to user).
switch get_param(gcb, 'rateEq')
    case 'k = A * exp( E / (RT) )'
        vis{index_coeff_Tk} = 'off';
        set_param(gcb, 'useTk','off');

    otherwise
        vis{index_coeff_Tk} = 'on';
        set_param(gcb, 'useTk','on');
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

coeff_a callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'coeff_b'))
        index_coeff_b = i;
    elseif (strcmp(names(i),'coeff_c'))
        index_coeff_c = i;

    end
end

% If there is no second reactant, don't offer a third reactant & don't show
% second & third reactant property choices
switch get_param(gcb, 'coeff_a')
    case '0'
        vis{index_coeff_b} = 'off';
        vis{index_coeff_c} = 'off';

    otherwise
        vis{index_coeff_b} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

coeff_b callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'coeff_c'))
```

```
        index_coeff_c = i;
    elseif (strcmp(names(i),'CBinit'))
        index_CBinit = i;
    elseif (strcmp(names(i),'CCinit'))
        index_CCinit = i;
    end
end

% If there is no second reactant, don't offer a third reactant
switch get_param(gcb, 'coeff_b')
    case '0'
        vis{index_coeff_c}= 'off';
        vis{index_CBinit} = 'off';
        vis{index_CCinit} = 'off';

    otherwise
        vis{index_coeff_c}= 'on';
        vis{index_CBinit} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

coeff_c callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'CCinit'))
        index_CCinit = i;
    end
end

% If there is no third reactant, don't offer a third reactant
% initialization
switch get_param(gcb, 'coeff_c')
    case '0'
        %vis{index_CCinit} = 'off';

    otherwise
        vis{index_CCinit} = 'on';
end
```

```
% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

coeff_d callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'coeff_e'))
        index_coeff_e = i;
    elseif (strcmp(names(i),'coeff_f'))
        index_coeff_f = i;
    elseif (strcmp(names(i),'prod_p2'))
        index_prod_p2 = i;
    elseif (strcmp(names(i),'deltaH_p2'))
        index_deltaH_p2 = i;
    elseif (strcmp(names(i),'Cp_p2'))
        index_Cp_p2 = i;
    elseif (strcmp(names(i),'prod_p3'))
        index_prod_p3 = i;
    elseif (strcmp(names(i),'deltaH_p3'))
        index_deltaH_p3 = i;
    elseif (strcmp(names(i),'Cp_p3'))
        index_Cp_p3 = i;

    end
end

% If there is no second product, don't offer a third product & don't show
% second & third product property choices
switch get_param(gcb, 'coeff_d')
    case '0'
        vis{index_coeff_e} = 'off';
        vis{index_coeff_f} = 'off';
        vis{index_prod_p2} = 'off';
        vis{index_deltaH_p2} = 'off';
        vis{index_Cp_p2} = 'off';
        vis{index_prod_p3} = 'off';
        vis{index_deltaH_p3} = 'off';
        vis{index_Cp_p3} = 'off';

    otherwise
        vis{index_coeff_e} = 'on';
end
```

```
% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

coeff_e callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'coeff_f'))
        index_coeff_f = i;
    elseif (strcmp(names(i),'prod_p2'))
        index_prod_p2 = i;
    elseif (strcmp(names(i),'deltaH_p2'))
        index_deltaH_p2 = i;
    elseif (strcmp(names(i),'Cp_p2'))
        index_Cp_p2 = i;
    elseif (strcmp(names(i),'prod_p3'))
        index_prod_p3 = i;
    elseif (strcmp(names(i),'deltaH_p3'))
        index_deltaH_p3 = i;
    elseif (strcmp(names(i),'Cp_p3'))
        index_Cp_p3 = i;

    end
end


% If there is no second product, don't offer a third product & don't show
% second & third product property choices
switch get_param(gcb, 'coeff_e')
    case '0'
        vis{index_coeff_f} = 'off';
        vis{index_prod_p2} = 'off';
        vis{index_deltaH_p2} = 'off';
        vis{index_Cp_p2} = 'off';
        vis{index_prod_p3} = 'off';
        vis{index_deltaH_p3} = 'off';
        vis{index_Cp_p3} = 'off';
        coeff_f = 0;
        prod_deltaH_p2 = 0;
        Cp_p2 = 0;
        prod_deltaH_p3 = 0;
        Cp_p3 = 0;
```

```
    otherwise
        vis{index_coeff_f} = 'on';
        vis{index_prod_p2} = 'on';
        vis{index_deltaH_p2} = 'on';
        vis{index_Cp_p2} = 'on';

end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

coeff_f callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'prod_p3'))
        index_prod_p3 = i;
    elseif (strcmp(names(i),'deltaH_p3'))
        index_deltaH_p3 = i;
    elseif (strcmp(names(i),'Cp_p3'))
        index_Cp_p3 = i;

    end
end

% If there is no third product, don't show third product property choices
switch get_param(gcb, 'coeff_f')
    case '0'
        vis{index_prod_p3} = 'off';
        vis{index_deltaH_p3} = 'off';
        vis{index_Cp_p3} = 'off';

    otherwise
        vis{index_prod_p3} = 'on';
        vis{index_deltaH_p3} = 'on';
        vis{index_Cp_p3} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

nonElem callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'alpha'))
        index_alpha = i;
    elseif (strcmp(names(i),'beta'))
        index_beta = i;
    elseif (strcmp(names(i),'gamma'))
        index_gamma = i;
    end
end

% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'nonElem')
    case 'off'
        vis{index_alpha} = 'off';
        vis{index_beta} = 'off';
        vis{index_gamma} = 'off';

    otherwise
        vis{index_alpha} = 'on';
        vis{index_beta} = 'on';
        vis{index_gamma} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

nonElemrB callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);
```

```
% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'rBfactor'))
        index_rBfactor = i;
    end
end

% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'nonElemrB')
    case 'off'
        vis{index_rBfactor} = 'off';

    otherwise
        vis{index_rBfactor} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

nonElemrC callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'rCfactor'))
        index_rCfactor = i;
    end
end

% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'nonElemrC')
    case 'off'
        vis{index_rCfactor} = 'off';

    otherwise
        vis{index_rCfactor} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

nonElemrD callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'rDfactor'))
        index_rDfactor = i;
    end
end

% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'nonElemrD')
    case 'off'
        vis{index_rDfactor} = 'off';

    otherwise
        vis{index_rDfactor} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

nonElemrE callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'rEfactor'))
        index_rEfactor = i;
    end
end
```

```
% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'nonElemrE')
    case 'off'
        vis{index_rEfactor} = 'off';

    otherwise
        vis{index_rEfactor} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

nonElemrF callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'rFfactor'))
        index_rFfactor = i;
    end
end

% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'nonElemrF')
    case 'off'
        vis{index_rFfactor} = 'off';

    otherwise
        vis{index_rFfactor} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

prod_p1 callback

```
% choose when to have heat of formation and heat capacity be retrieved from
% a structure or let user enter the values

% collect enable values of each blue numbered mask parameter
en = get_param(gcb,'MaskEnables');
```

```
% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'deltaH_p1'))
        index_deltaH_p1 = i;
    elseif (strcmp(names(i),'Cp_p1'))
        index_Cp_p1 = i;
    elseif (strcmp(names(i),'rho_p1'))  % note: rho is not visible to user
because it is not used in any calculations
        index_rho_p1 = i;
    end
end

% different behavior depending on whether a user defined product or product
% from a stored library fluid
switch get_param(gcb, 'prod_p1')
    case 'Other (user defined)'
        % enable boxes so user can fill in
        en{index_deltaH_p1} = 'on';
        en{index_Cp_p1} = 'on';
        en{index_rho_p1} = 'on';

    otherwise
        % get values of fluid from saved structure
        FluidProp = eval(['Fluid.',get_param(gcb,'prod_p1')]);

        HeatCapD = FluidProp.Cp;
        set_param(gcb,'Cp_p1',num2str(HeatCapD));

        HeatForm_p1 = FluidProp.HeatForm;
        set_param(gcb,'deltaH_p1',num2str(HeatForm_p1));

        Density_p1 = FluidProp.Rho;
        set_param(gcb,'rho_p1',num2str(Density_p1));

        % unenable boxes so user can't fill in
        en{index_deltaH_p1} = 'off';
        en{index_Cp_p1} = 'off';
        en{index_rho_p1} = 'off';
end

% apply changes to which parameters are enabled
set_param(gcb,'MaskEnables',en)
```

prod_p2 callback

```
% choose when to have heat of formation and heat capacity be retrieved from
% a structure or let user enter the values
```

```
% collect enable values of each blue numbered mask parameter
en = get_param(gcb,'MaskEnables');

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'deltaH_p2'))
        index_deltaH_p2 = i;
    elseif (strcmp(names(i),'Cp_p2'))
        index_Cp_p2 = i;
    end
end

% different behavior depending on whether a user defined product or product
% from a stored library fluid
switch get_param(gcb, 'prod_p2')
    case 'Other (user defined)'
        % enable boxes so user can fill in
        en{index_deltaH_p2} = 'on';
        en{index_Cp_p2} = 'on';
    otherwise
        % get values of fluid from saved structure
        FluidProp = eval(['Fluid.',get_param(gcb,'prod_p2')]);
        Cp_p2 = FluidProp.Cp;
        set_param(gcb,'Cp_p2',num2str(Cp_p2));
        HeatForm_p2 = FluidProp.HeatForm;
        set_param(gcb,'deltaH_p2',num2str(HeatForm_p2));

        % unenable boxes so user can't fill in
        en{index_deltaH_p2} = 'off';
        en{index_Cp_p2} = 'off';
end

% apply changes to which parameters are enabled
set_param(gcb,'MaskEnables',en)
```

prod_p3 callback

```
% choose when to have heat of formation and heat capacity be retrieved from
% a structure or let user enter the values

% collect enable values of each blue numbered mask parameter
en = get_param(gcb,'MaskEnables');

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'deltaH_p3'))
        index_deltaH_p3 = i;
    elseif (strcmp(names(i),'Cp_p3'))
        index_Cp_p3 = i;
    end
end

% different behavior depending on whether a user defined product or product
```

```
% from a stored library fluid
switch get_param(gcb, 'prod_p3')
    case 'Other (user defined)'
        % enable boxes so user can fill in
        en{index_deltaH_p3} = 'on';
        en{index_Cp_p3} = 'on';
    otherwise
        % get values of fluid from saved structure
        FluidProp = eval(['Fluid.',get_param(gcb,'prod_p3')]);
        Cp_p3 = FluidProp.Cp;
        set_param(gcb,'Cp_p3',num2str(Cp_p3));
        HeatForm_p3 = FluidProp.HeatForm;
        set_param(gcb,'deltaH_p3',num2str(HeatForm_p3));

        % unenable boxes so user can't fill in
        en{index_deltaH_p3} = 'off';
        en{index_Cp_p3} = 'off';
end

% apply changes to which parameters are enabled
set_param(gcb,'MaskEnables',en)
```

newRho callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'rho'))
        index_rho = i;
    end
end

rho = get_param(gcb, 'newRho');

% If user wants to enter a new Rho, provide space
switch get_param(gcb, 'newRho')
    case 'off'
        vis{index_rho} = 'off';

    otherwise
        vis{index_rho} = 'on';
end
```

```
% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

newCps callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'userCps'))
        index_userCps = i;
    end
end

userCps = get_param(gcb, 'newCps');

% If user wants to enter a new Cps, provide space
switch get_param(gcb, 'newCps')
    case 'off'
        vis{index_userCps} = 'off';

    otherwise
        vis{index_userCps} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

newdeltaHform callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'userdeltaHform'))
```

```
        index_userdeltaHform = i;
    end
end

% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'newdeltaHform')
    case 'off'
        vis{index_userdeltaHform} = 'off';

    otherwise
        vis{index_userdeltaHform} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

newdeltaCp callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'userdeltaCp'))
        index_userdeltaCp = i;
    end
end

% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'newdeltaCp')
    case 'off'
        vis{index_userdeltaCp} = 'off';

    otherwise
        vis{index_userdeltaCp} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

newdeltaHrxn callback

```
% choose what options to make visible
```

```
% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'userdeltaHrxn'))
        index_userdeltaHrxn = i;
    end
end

% If user wants to enter a new deltaCp, provide space and hide
% calculated deltaCp
switch get_param(gcb, 'newdeltaHrxn')
    case 'off'
        vis{index_userdeltaHrxn} = 'off';

    otherwise
        vis{index_userdeltaHrxn} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

A.1.3 Subsystem

A.1.3.1 Structure

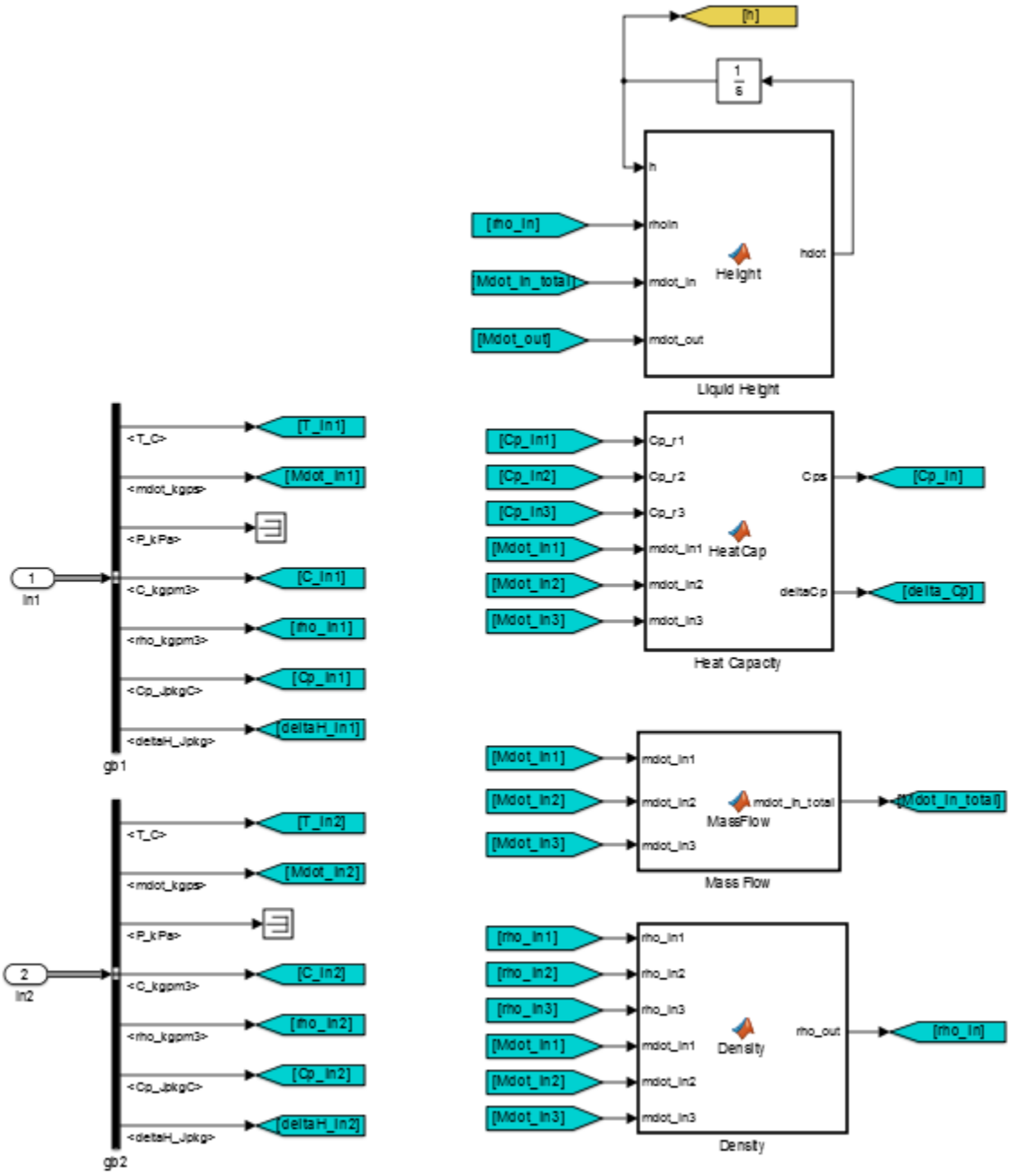

Figure A.4 CSTR subsystem structure (1/4).

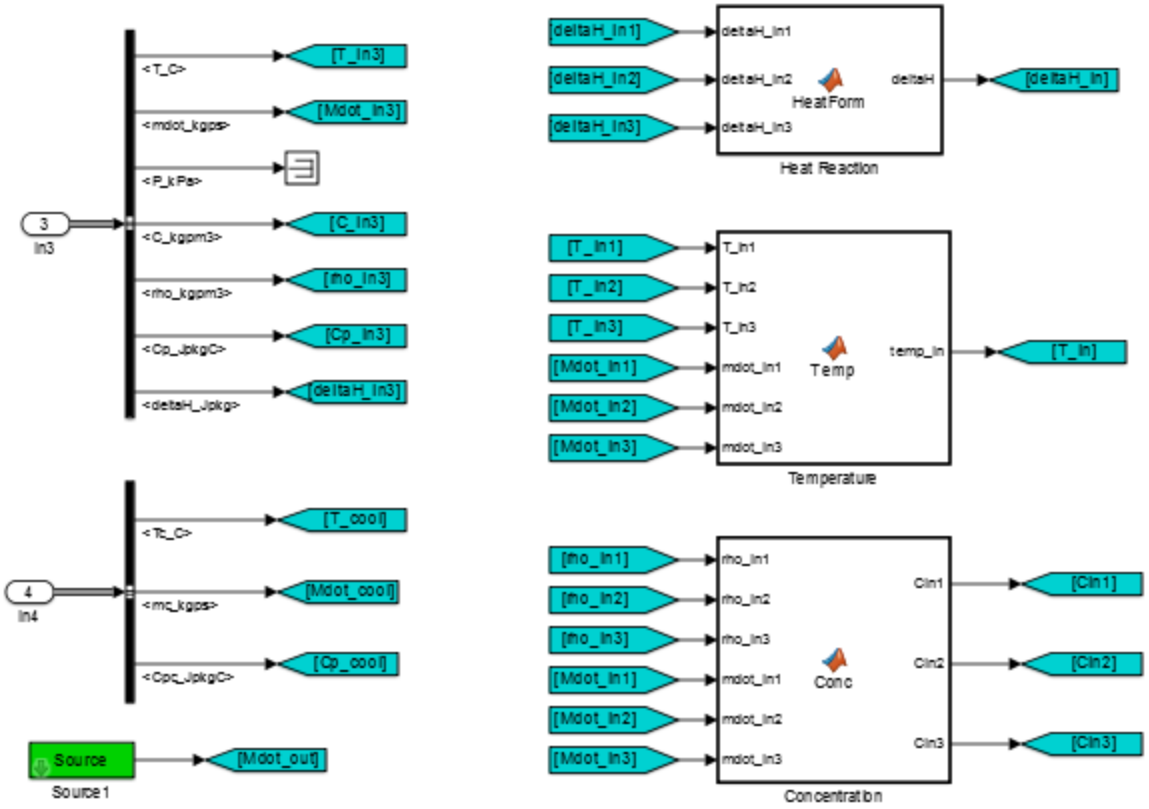

Figure A.5 CSTR subsystem structure (2/4).

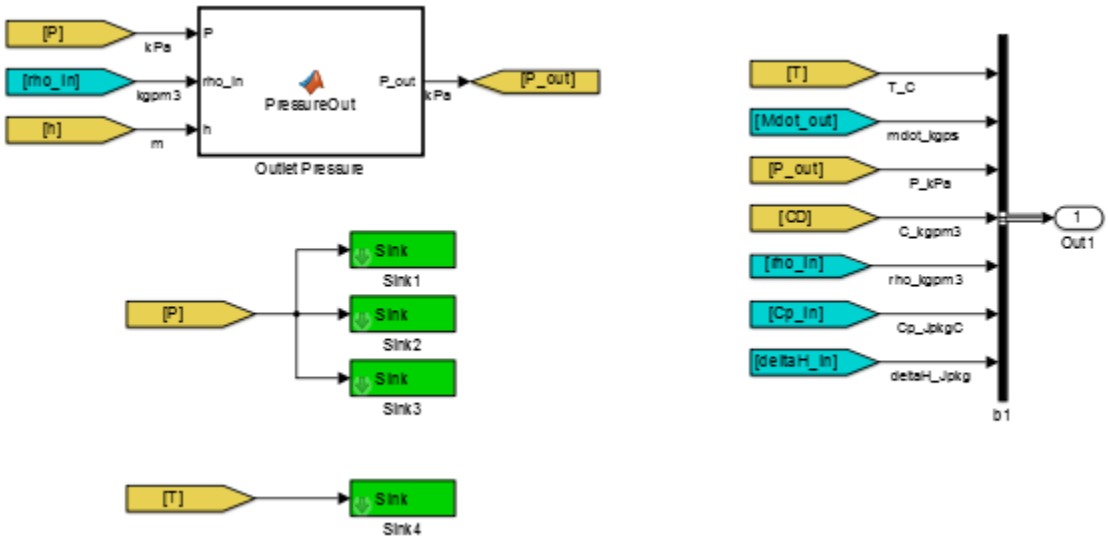

Figure A.6 CSTR subsystem structure (3/4).

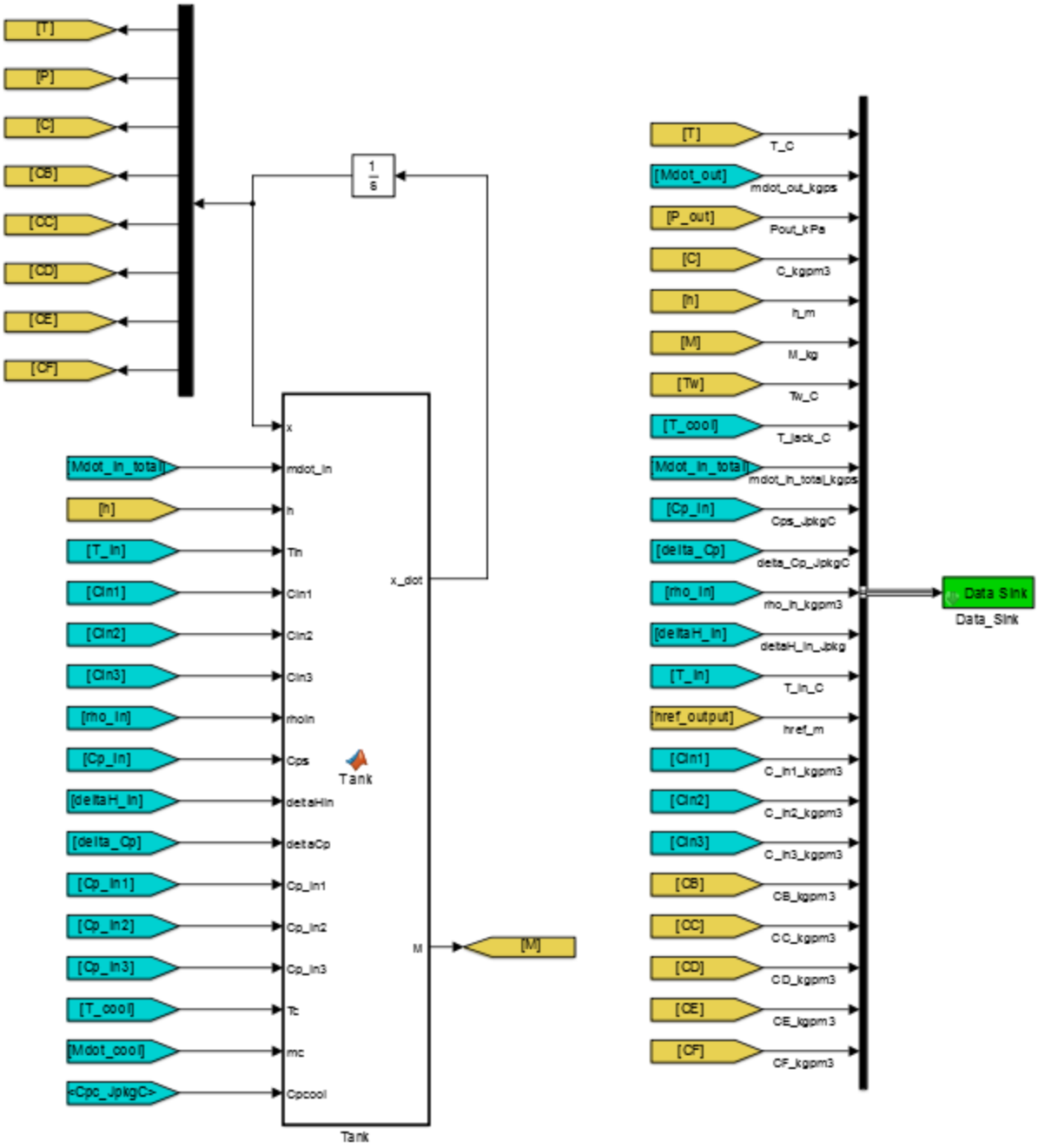

Figure A.7 CSTR subsystem structure (4/4).

A.1.3.2 Matlab Functions

hdot Matlab function

```
% Calculate solution heat capacity and change in heat capacity
```

```
function hdot    = Height(d, ht, h, rhoin,mdot_in, mdot_out)
%{
INPUTS
rhoin : density of fluid (either flow in or user defined density) (kg/m^3)
mdot_in : total mass flow rate in (kg/s)
mdot_out : total mass flow rate out (kg/s)

PARAMETERS (from mask):
d : tank diameter (m)
ht : tank height (m)
h : liquid height (m)

OUTPUTS:
hdot : derivative of liquid height (m/s)
%}
%% Calculate change in liquid height in tank

% Geometric calculations
A_cross = pi*d^2/4;     % cross-sectional area of tank (m^2)

% Outputs
% Liquid height in tank (m)
% Error check height - can't be less than zero or greater than tank height
if (h<0)
    hdot = 0;
elseif (h>ht)
    hdot = 0;
else
    hdot = ((mdot_in - mdot_out)/(rhoin*A_cross));      % height derivative
end
```

HeatCap Matlab function

```
% Calculate solution heat capacity and change in heat capacity

function [Cps, deltaCp] = HeatCap(coeff_a, coeff_b, coeff_c, coeff_d, ...
    coeff_e, coeff_f, Cp_p1, Cp_p2, Cp_p3, Cp_r1, Cp_r2, Cp_r3,...
    mdot_in1, mdot_in2, mdot_in3, newdeltaCp, userdeltaCp, newCps, userCps)

%% Calculate change in heat capacity

% Check if user entered change in heat capacity
if (newdeltaCp)
    deltaCp = userdeltaCp;
else  % Calculate change in heat capacity

    % error checking
    if (mdot_in2 == 0)      % if there is no second mass flow rate
        coeff_b = 0;        % then there is no second reactant
    end

    if (mdot_in3 == 0)      % if there is no third mass flow rate
```

```
        coeff_c = 0;        % then there is no third reactant
    end

    % ensure that won't divide by zero
    if (coeff_a > 0)
        % Calculate change in heat capacity [1]
        deltaCp = (coeff_d/coeff_a)*Cp_p1 + (coeff_e/coeff_a)*Cp_p2 + ...
            (coeff_f/coeff_a)*Cp_p3 - (coeff_c/coeff_a)*Cp_r3 - ...
            (coeff_b/coeff_a)*Cp_r2 - Cp_r1;
    else
        deltaCp = 0;
    end

end


%% Calculate heat capacity of solution

% Check if user entered heat capacity of solution
if (newCps)
    Cps = userCps;
else  % Calculate heat capacity of solution
    % Calculate theta parameter [2]
    thetaA = mdot_in1 / mdot_in1;
    thetaB = mdot_in2 / mdot_in1;
    thetaC = mdot_in3 / mdot_in1;
    thetaD = 0;         % no product flow in
    thetaE = 0;         % no product flow in
    thetaF = 0;         % no product flow in
    % Calculate heat capacity of solution [3]
    Cps = thetaA*Cp_r1 + thetaB*Cp_r2 + thetaC*Cp_r3 + thetaD*Cp_p1 + ...
        thetaE*Cp_p2 + thetaF*Cp_p3;
end

%%
%{
References:
[1] Element of Chemical Reaction Engineering, Second Edition. H. Scott
Fogler. Page 391, equation (8-25).
[2] Element of Chemical Reaction Engineering, Second Edition. H. Scott
Fogler. Page 80.
[3] Element of Chemical Reaction Engineering, Second Edition. H. Scott
Fogler. Page 431.
%}
```

MassFlow Matlab function

```
% Calculate total mass flow rate in

function mdot_in_total  = MassFlow(mdot_in1, mdot_in2, mdot_in3)

% sum all mass flows in
mdot_in_total = mdot_in1 + mdot_in2 + mdot_in3;
```

Density Matlab function

```
% Calculate the density based on inlet flow densities and rates

function rho_out  = Density(rho_in1, rho_in2, rho_in3, mdot_in1, ...
    mdot_in2, mdot_in3, newRho, rho)

% Check if user entered density
if (newRho)
    rho_out = rho;

% Calculate rho_out, density of solution
else
    % calculate volumetric flow rate of each inlet flow
    v_in1 = mdot_in1 / rho_in1;
    v_in2 = mdot_in2 / rho_in2;
    v_in3 = mdot_in3 / rho_in3;

    % sum total volumetric flow rate of inlets
    v_in_total = v_in1 + v_in2 + v_in3;

    % calculate density of solution
    rho_out = (v_in1*rho_in1 + v_in2*rho_in2 + v_in3*rho_in3) / ...
        v_in_total;

    % this is calculating the density of solution based on average of inlet
    % densities using the volumetric flow rates of each flow
end
```

HeatForm Matlab function

```
% Calculate the heat of reaction (delta heat of formation / enthalpy of
% formation)

function deltaH = HeatForm(deltaH_in1, deltaH_in2, deltaH_in3,coeff_a,...
    coeff_b, coeff_c, coeff_d, coeff_e, coeff_f, deltaH_p1, deltaH_p2,...
    deltaH_p3, newdeltaHform, userdeltaHform)

% Check if user entered heat of reaction
if (newdeltaHform)
    deltaH = userdeltaHform;

% Calculate the heat of reaction
else
    % error checking: ensure reactants & products that aren't being used
    % are not included in the heat of formation calculation
    if (coeff_e == 0)       % if there is no second product
        coeff_f = 0;        % then there is no third product
    end
    if (coeff_b == 0)       % if there is no second reactant
        coeff_c = 0;        % then there is no third reactant
    end
```

```
    % ensure that won't divide by zero
    if (coeff_a > 0)
        % Calculate heat of reaction [1]
        deltaH = (coeff_d/coeff_a)*deltaH_p1 + (coeff_e/coeff_a)*deltaH_p2 +
...
            (coeff_f/coeff_a)*deltaH_p3 - (coeff_c/coeff_a)*deltaH_in3 - ...
            (coeff_b/coeff_a)*deltaH_in2 - deltaH_in1;
    else
        deltaH = 0;
    end

end

%{
Reference:
[1] Element of Chemical Reaction Engineering, Second Edition. H. Scott
Folger. Page 391, equation (8-24).
%}
```

Temp Matlab function

```
% Calculate the temperature of the inlet flow based on inlet flow
% temperature and mass flow rates

function temp_in  = Temp(T_in1, T_in2, T_in3, mdot_in1, mdot_in2, ...
    mdot_in3, coeff_b, coeff_c)

% error checking: ensure that if reactants that aren't being used are not
% included in the temperature calculation
if (coeff_b == 0)       % if there is no second reactant
    mdot_in2 = 0;       % then there is no second mass flow in
    coeff_c = 0;        % then there is no third reactant
end
if (coeff_c == 0)       % if there is no third reactant
    mdot_in3 = 0;       % then there is no third mass flow in
end

% Calculate temp_in, temperature of combined flows into CSTR
% assuming there is no immediate temperature rise upon mixing feedstreams

% sum total mass flow in
mdot_total = mdot_in1 + mdot_in2 + mdot_in3;

% calculate temp_in - average temperature
temp_in = (mdot_in1*T_in1 + mdot_in2*T_in2 + mdot_in3*T_in3)/ (mdot_total);
```

Conc Matlab function

```
% Calculate the concentration of each inlet flow

function [Cin1, Cin2, Cin3]  = Conc(rho_in1, rho_in2, rho_in3,...
    mdot_in1, mdot_in2, mdot_in3)

% Calculate volumetric flow rates
```

```
v0_1 = mdot_in1 / rho_in1;
v0_2 = mdot_in2 / rho_in2;
v0_3 = mdot_in3 / rho_in3;

% Calculate total volumetric flow rate
v0_total = v0_1 + v0_2 + v0_3;

% Calculate concentration of reactant 1 [1]
Cin1 = mdot_in1 / v0_total;

% Calculate concentration of reactant 2
Cin2 = mdot_in2 / v0_total;

% Calculate concentration of reactant 3
Cin3 = mdot_in3 / v0_total;

%{
Reference:
[1] Element of Chemical Reaction Engineering, Second Edition. H. Scott
Folger. Page 79, equation (3-27).
*Example also shown in pages 402-403
%}
```

PressureOut Matlab function

```
% Calculate the algebraic pressure at the outlet
% Assumption: Outlet is at the bottom of the tank such that the static
% pressure is calculated by the pressure in the tank plus the hydrostatic
% pressure

function P_out = PressureOut(P, rho_in, h)
% P at top of reactor, [kPa] (Pa = kg/(m*s^2))
% rho_in, [kg/m^3]
% h liquid height, [m]

g = 9.80665;                    % m/s^2

% Calculate pressure at outlet in kPa
P_out = P + (1/1000)*rho_in*g*h; % kPa

end
```

Tank Matlab function

```
function [x_dot, M]= Tank(x, mdot_in, h, Tin, ...
    Cin1, Cin2, Cin3, rhoin, Cps, deltaHin, deltaCp, Cp_in1, Cp_in2, ...
    Cp_in3, d, At, Ut, Ak, E, Tk, useTk, coeff_a, coeff_b, coeff_c,...
    coeff_d, coeff_e, coeff_f, alpha, beta, gamma, nonElem, Cp_p1, Cp_p2,...
    Cp_p3, nonElemrB, nonElemrC, nonElemrD, nonElemrE, nonElemrF,
rBfactor,...
    rCfactor, rDfactor, rEfactor, rFfactor, Tc, mc, Cpcool)
%{
OUTPUTS:
x_dot : [temperature in tank derivative (C/s), pressure in tank derivative
```

```
(N/m^2s), concentration derivative (kg/m^3s),
liquid height derivative (m/s), concentration B derivative (kg/m^3s),
concentration C derivative (kg/m^3s), concentration D derivative (kg/m^3s),
 concentration E derivative (kg/m^3s), concentration F derviative (kg/m^3s)]
M : mass of liquid in tank (kg)

INPUTS:
x : temperature in tank (C), pressure at bottom of tank (kPa),
concentration of reactant A/1 (kg/m^3),
concentration of reactant B/2 (kg/m^3), conc C/3 (kg/m^3),
conc D/4 (kg/m^3), conc E/5 (kg/m^3), conc F/6 (kg/m^3);
x = [T, P, C, CB, CC, CD, CE, CF]
mdot_in : total mass flow rate in to tank (kg/s)
h: liquid height (m)
Tin : temperature of inlet flow (average temperature) (C)
Cin1: concentration of reactant 1 in combined flow of up to 3 inlets (kg/m^3)
Cin2: concentration of reactant 2 in combined flow of up to 3 inlets (kg/m^3)
Cin3: concentration of reactant 3 in combined flow of up to 3 inlets (kg/m^3)
rhoin : density of fluid (either flow in or user defined density) (kg/m^3)
Cps : heat capacity of the solution (either calculated or user defined)
(J/kgC)
deltaHin : heat of reaction (delta heat of formation) (J/kgC)
deltaCp : change in heat capacity per mole of A reacted (J/kgC)
Cp_in1 : specific heat of reactant 1 (J/kgC)
Cp_in2 : specific heat of reactant 2 (J/kgC)
Cp_in3 : specific heat of reactant 3 (J/kgC)
Tc : coolant temperature (C)
mc : coolant mass flow rate (kg/s)
Cpcool : coolant heat capacity (J/kgC)

PARAMETERS (from mask):
d : tank diameter (m)
At : tank surface area (m^2)
Ut : heat transfer coefficient of tank (J/s m^2 C)
Ak : pre-exponential factor (1/s)
E : activation energy (J/mol)
Tk : temperature in rate equaiton (C)
useTk : binary variable to select whether rate law uses Tk (1 or 0) (-)
coeff_a : reactant coefficient for reactant A (-)
coeff_b : reactant coefficient for reactant B (-)
coeff_c : reactant coefficient for reactant C (-)
coeff_d : reactant coefficient for product D (-)
coeff_e : reactant coefficient for product E (-)
coeff_f : reactant coefficient for product F (-)
alpha : non-elementary rate law power for reactant A (-)
beta : non-elementary rate law power for reactant B (-)
gamma : non-elementary rate law power for reactant C (-)
nonElem : binary variable to select whether rate law is nonelementary (1 or
0, 1 is nonelementary)
or elementary (0) (-)
Cp_p1 : heat capacity of product 1 (J/kgC)
Cp_p2 : heat capacity of product 2 (J/kgC)
Cp_p3 : heat capacity of product 3 (J/kgC)
rBfactor : multiplicative factor for rB rate law (-)
rCfactor : multiplicative factor for rC rate law (-)
```

```
rDfactor : multiplicative factor for rD rate law (-)
rEfactor : multiplicative factor for rE rate law (-)
rFfactor : multiplicative factor for rF rate law (-)
nonElemrB : binary variable to select whether rB rate law is nonelementary (1
or 0, 1 is nonelementary)
nonElemrC : binary variable to select whether rC rate law is nonelementary (1
or 0, 1 is nonelementary)
nonElemrD : binary variable to select whether rD rate law is nonelementary (1
or 0, 1 is nonelementary)
nonElemrE : binary variable to select whether rE rate law is nonelementary (1
or 0, 1 is nonelementary)
nonElemrF : binary variable to select whether rF rate law is nonelementary (1
or 0, 1 is nonelementary)

Other available parameters from mask:
ht : tank height (m)
t : tank wall thickness (m)
Vt : tank volume (m^3)
deltaH_p1 : heat of formation of product 1 (J/kg)
deltaH_p2 : heat of formation of product 2 (J/kg)
deltaH_p3 : heat of formation of product 3 (J/kg)
userCps : binary variable to indicate whether use entered own Cps value (-)
%}

% Parse states
T   = x(1);             % Temperature in the tank (C)
P   = x(2);             % Pressure at bottom of the tank (kPa)
C   = x(3);             % Concentration of reactant A/1 (kg/m^3)

C_B = x(4);             % Concentration of reactant B/2 (kg/m^3)
C_C = x(5);             % Concentration of reactant C/3 (kg/m^3)
C_D = x(6);             % Concentration of product D/4 (kg/m^3)
C_E = x(7);             % Concentration of product G/5 (kg/m^3)
C_F = x(8);             % Concentration of product F/6 (kg/m^3)

%% Inlet flow calculations
v0  = mdot_in / rhoin;  % total volumetric flow rate in (m^3/s)

F01  = Cin1 * v0;       % mass flow rate in of reactant 1 (kg/s)
F02  = Cin2 * v0;       % mass flow rate in of reactant 2 (kg/s)
F03  = Cin3 * v0;       % mass flow rate in of reactant 3 (kg/s)

F0  = F01 + F02 + F03;  % mass flow rate in of all reactants (kg/s)

%% Geometric calculations
A_cross = pi*d^2/4;     % cross-sectional area of tank (m^2)
V       = A_cross * h;  % volume of liquid in tank (m^3)

%% Rate law calculations
% Rate law options - handle 2 possible rate laws from user
% Note: Need to convert temperatures given in Celsius by user to Kelvin
% (then Rankine) to make units work with R given as J/(mol R)
if (useTk)
    Trate =( (1/(Tk+273.15)) - (1/(T+273.15)) );    % K
```

```
else
    Trate = (1/(T+273.15));                          % K
end

R = 4.621762000000000;  % gas constant, J/(mol*R)

% rate constant
k = Ak*exp(((-E*Trate)/(R*(9/5))));                    % 1/s, 9/5 to convert
from K to R to merge with R units of J/(mol*R)

% Rate law
if (nonElem)
    % non-elementary rate law using alpha, beta, gamma provided by user in
    % mask
    rA = -k * C^alpha * C_B^beta * C_C^gamma;
else
    % rate law assuming an elementary reaction rate using a, b, c provided
    % by user in mask [4]
    rA = -k * C^coeff_a * C_B^coeff_b * C_C^coeff_c;
end

% calculate other rate laws based on rA, cofficients, and nonElemrX values
[5]
if (nonElemrB)
    rB = rBfactor * (coeff_b / coeff_a) * rA;    % nonelementary rB rate law -
this is needed for when converting from concentration units in lbmol.
    % rBfactor = MWb/MWa
else
    rB = (coeff_b / coeff_a) * rA;               % elementary rB rate law
end
if (nonElemrC)
    rC = rCfactor * (coeff_c / coeff_a) * rA;    % nonelementary rC rate law
else
    rC = (coeff_c / coeff_a) * rA;               % elementary rC rate law
end
if (nonElemrD)
    rD = -rDfactor * (coeff_d / coeff_a) * rA;
else
    rD = -(coeff_d / coeff_a) * rA;
end
if (nonElemrE)
    rE = -rEfactor * (coeff_e / coeff_a) * rA;
else
    rE = -(coeff_e / coeff_a) * rA;
end
if (nonElemrF)
    rF = -rFfactor * (coeff_f / coeff_a) * rA;
else
    rF = -(coeff_f / coeff_a) * rA;
end

%% Calculate OUTPUTS

% Error checking
```

```
if (coeff_e == 0)
    coeff_f = 0;
    Cp_p2 = 0;
end
if (coeff_f == 0)
    Cp_p3 = 0;
end

% Heat of reaction calculation
Tr_K  = 293.15;          % reference temperature (K)
Tr = Tr_K - 273.15;      % reference temperature (C)
deltaHRX = deltaHin+deltaCp*(T-Tr);     % Heat of reaction J/kg [6]

% Assumption
W_dot   = 0;

% Calculate state derivatives
% Tank temperature (C) [1]

% Heat transfer from coolant jacket
Qnew = mc*Cpcool*(Tc-T)*(1-exp((-Ut*At)/(mc*Cpcool)));

num = (Qnew)-W_dot-(F01*Cps*(T-Tin))+((deltaHRX)*(rA*V));

denom = V*(C*Cp_in1+C_B*Cp_in2+C_C*Cp_in3+C_D*Cp_p1+ C_E*Cp_p2+ C_F*Cp_p3);
%J/C

T_dot = (num)/(denom);

% Tank pressure (kPa) - assume no pressure change because a very small
% change in liquid volume
P_dot = 0;
% Reactant A concentration (kg/m^3) [2]
C_dot = (v0/V)*(Cin1-C) + rA;

% Extra concentrations (kg/m^3) - equations from examples
C_B_dot = (v0/V)*(Cin2-C_B) + rB;
C_C_dot = (v0/V)*(Cin3-C_C) + rC;
C_D_dot = (v0/V)*(0-C_D) + rD;
C_E_dot = (v0/V)*(0-C_E) + rE;
C_F_dot = (v0/V)*(0-C_F) + rF;

% Mux state derivatives
x_dot = [T_dot, P_dot, C_dot, C_B_dot, C_C_dot, C_D_dot,...
    C_E_dot, C_F_dot];

% Other outputs
M = rhoin * V;              % tank mass (kg)

end

%{
References:
```

```
[1] Process Dyanmics and Control 1st edition by Dale Seborg, page 28,
equation (2-31)

[1] Process Dyanmics and Control 1st edition by Dale Seborg, page 28,
equation (2-30)

[3] Process Dyanmics and Control 1st edition by Dale Seborg, page 24,
equation (2-19)

[4] Elements of Chemical Reaction Engineering 2nd edition by H. Scott
Fogler, page 66

[4] Elements of Chemical Reaction Engineering 2nd edition by H. Scott
Fogler, page 73, equation (3-22)

[6] Elements of Chemical Reaction Engineering 2nd edition by H. Scott
Fogler, page 402, equation (8-27)
%}
```

A.2 Continuous Stirred Tank Reactor Jacket

A.2.1 Initialization

```
% load fluid properties
load('Fluids.mat');
```

A.2.2 Parameters and Callbacks

Figure A.8 shows the CSTR jacket mask parameters.

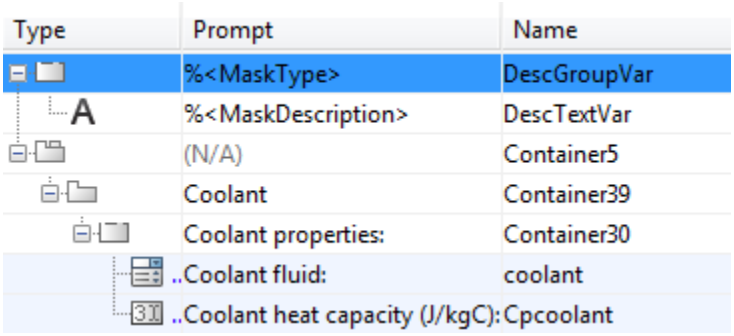

Figure A.8 CSTR jacket mask parameters.

The CSTR jacket has the following callback associated with the coolant parameter.

coolant callback

```
% choose when to have heat capacity be retrieved from
% a structure or let user enter value

% collect enable values of each blue numbered mask parameter
en = get_param(gcb,'MaskEnables');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'Cpcoolant'))
        index_Cpcoolant = i;
    end
end

% different behavior depending on whether a user defined product or product
% from a stored library fluid
switch get_param(gcb, 'coolant')
    case 'Other (user defined)'
        % enable boxes so user can fill in
        en{index_Cpcoolant} = 'on';

    otherwise
        % get values of fluid from saved structure
        FluidProp = eval(['Fluid.',get_param(gcb,'coolant')]);

        Cpcoolant = FluidProp.Cp;
        set_param(gcb,'Cpcoolant',num2str(Cpcoolant));

        % unenable boxes so user can't fill in
        en{index_Cpcoolant} = 'off';
end

% apply changes to which parameters are enabled
set_param(gcb,'MaskEnables',en)
```

A.2.3 Subsystem

A.2.3.1 Structure

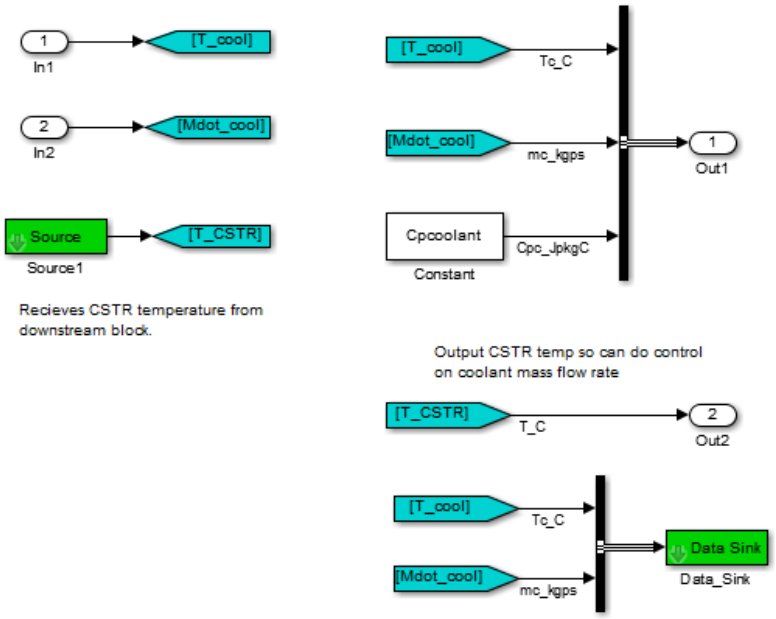

Figure A.9 CSTR jacket subsystem structure.

A.2.3.2 Matlab Functions

The CSTR jacket subsystem has no Matlab functions.

A.3 Pump

A.3.1 Initialization

```
load('Fluids.mat');

fld = 'Water';
Fluid2 = eval(['Fluid.',fld]);
```

A.3.2 Parameters and Callbacks

Figure A.10 shows the pump mask parameters.

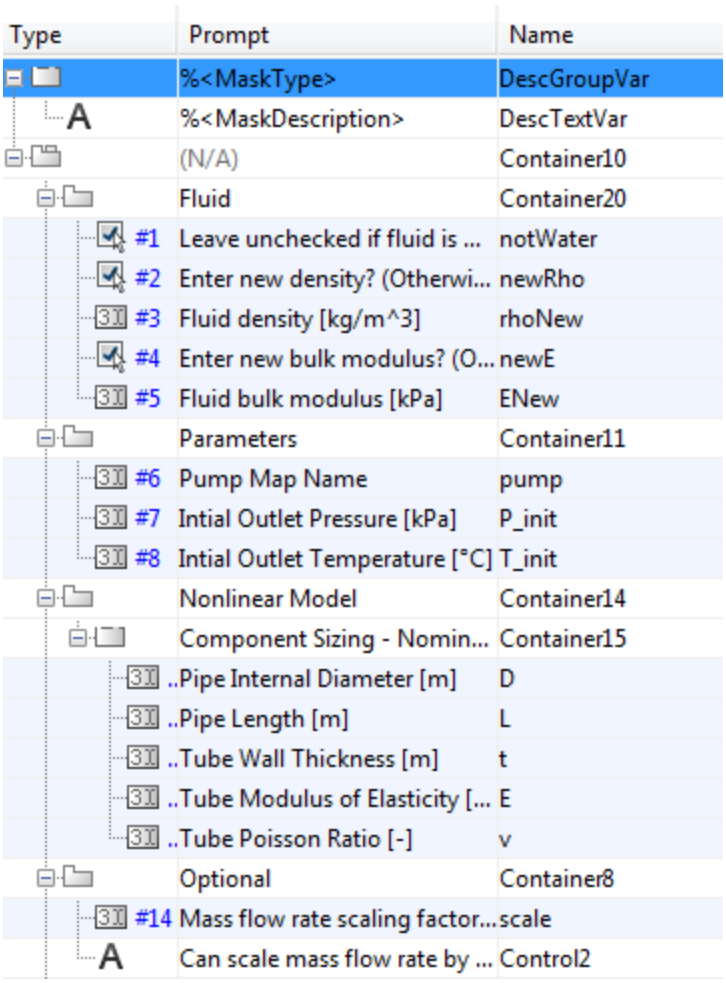

Figure A.10 Pump mask parameters.

The pump has the following callbacks associated with the parameters.

notWater callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'newRho'))
```

```
        index_newRho = i;
    elseif (strcmp(names(i),'newE'))
        index_newE = i;
    elseif (strcmp(names(i),'rhoNew'))
        index_rhoNew = i;
    elseif (strcmp(names(i),'ENew'))
        index_ENew = i;
    end
end

notWater = get_param(gcb, 'notWater');

% If user wants to enter a new Rho, provide space
switch get_param(gcb, 'notWater')
    case 'off'
        vis{index_newRho} = 'off';
        vis{index_newE} = 'off';
        vis{index_rhoNew} = 'off';
        vis{index_ENew} = 'off';
    otherwise   %'on'
        vis{index_newRho} = 'on';
        vis{index_newE} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);

fld = 'Water';
Fluid2 = eval(['Fluid.',fld]);
```

newRho callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'rhoNew'))
        index_rhoNew = i;
    end
end

rhoNew = get_param(gcb, 'rhoNew');

% If user wants to enter a new Rho, provide space
```

```
switch get_param(gcb, 'newRho')
    case 'off'
        vis{index_rhoNew} = 'off';

    otherwise
        vis{index_rhoNew} = 'on';

end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

newE callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'ENew'))
        index_ENew = i;
    end
end

ENew = get_param(gcb, 'ENew');

% If user wants to enter a new Rho, provide space
switch get_param(gcb, 'newE')
    case 'off'
        vis{index_ENew} = 'off';

    otherwise
        vis{index_ENew} = 'on';

end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

A.3.3 Subsystem

A.3.3.1 Structure

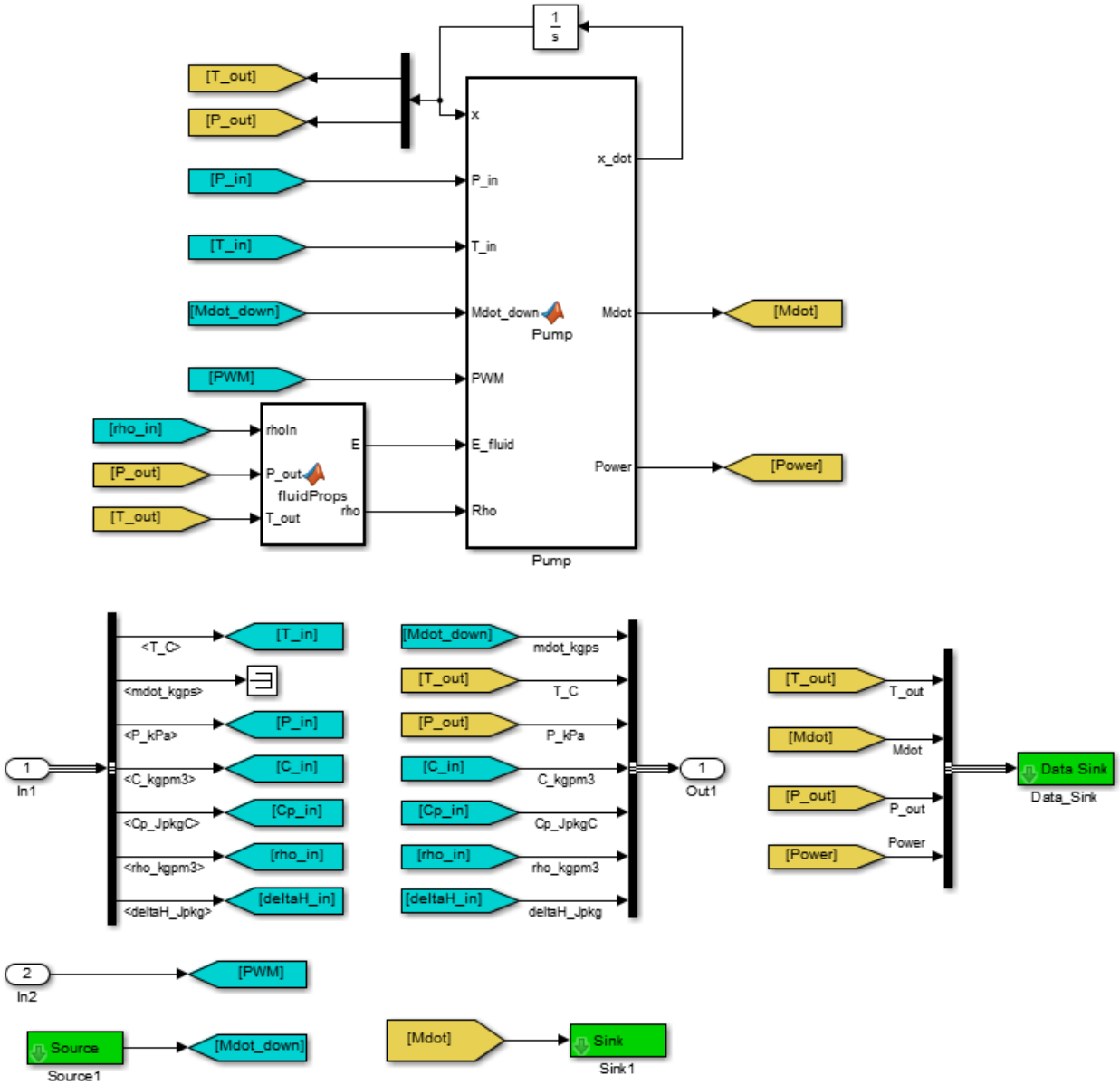

Figure A.11 Pump subsystem structure.

A.3.3.2 Matlab Functions

fluidProps Matlab function

```
function [E, rho] = fluidProps(newRho, newE, rhoIn, rhoNew, ENew,...
```

```
    P_out, T_out, notWater, Fluid2)

%{
  Calculate E and rho for fluid

  Since there are property tables for water, if fluid is water, then use
  property tables to look up density based on temperature and pressure.

  If it is not water, then probably use incoming fluid density (constant)
  regardless of temprature and pressure.

  Similarly, look up bulk modulus for water or enter own bulk modulus
  (water is used as default, if user doesn't want to enter a bulk modulus).

  **If other property tables are known, then incorporate those.
%}

% if not water (anything else
if (notWater)
   if (newE);
        E = ENew;                    % if user gives E value, use it
   else
        E = Fluid2.Bulk_Mod;                 % otherwise, use water E value
   end

   if (newRho)
       rho = rhoNew;
   else
       rho = rhoIn;
   end

% if Water
else
    % Check that P and T values are within range
    if (P_out < 500)
        P_out = 500;
    elseif (P_out > 20960.8)
        P_out = 20960.8;
    else
    end

    if (T_out < 2)
        T_out = 2;
    elseif (T_out > 500)
        T_out = 500;
    else
    end

    rho = qminterp2(Fluid2.T, Fluid2.P, Fluid2.Rho_pt, T_out, P_out);   %
faster than interp2

    E = Fluid2.Bulk_Mod;
end
```

Pump Matlab function

```
function [x_dot,Mdot,Power]= Pump(x,P_in,T_in,Mdot_down,PWM,E_fluid,Rho,...
    D,L,t,E,v,scale)

% Constants and Conversions
V           = L*pi*D^2/4;       % Volume
Rho_pump    = 1041;             % Fluid density used for pump head calc
g           = 9.81;             % Gravitational acceleration
A           = 1.27e-4;          % Area used for pump head calc
H_k1        = -0.0767;          % Pump head map coefficient (Bias [m])
H_k2        = 9.33e-2;          % Pump head map coefficient (dP [kPa])
H_k3        = 0.530;            % Pump head map coefficient (PWM [0-1])
Eta_k1      = 6.717;            % Pump efficiency map coefficient (Bias [0-
1])
Eta_k2      = 18.75;            % Pump efficiency map coefficient (WHP [W])
Eta_k3      = -26.92;           % Pump efficiency map coefficient (PWM [0-1])
Eta_k4      = -0.0174;          % Pump efficiency map coefficient (WHP^2 [0-
1])
Eta_k5      = 23;               % Pump efficiency map coefficient (PWM^2 [0-
1])
Eta_k6      = -22.89;           % Pump efficiency map coefficient (WHP*PWM
[0-1])

% Parse states
T       = x(1);                 % Temperature of fluid in pump
P       = x(2);                 % Pressure at outlet of pump

% Additional intermediate variable calculations
dP      = (P-P_in);             % Pressure difference [kPa], P_out - P_in
PWM     = PWM/100;              % PWM in duty ratio [0-1]
H       = H_k1 + H_k2*dP + H_k3*PWM;    % Pump head [m]
Q       = sqrt_Lip( max(0,2*g*(H - 1000*dP/(Rho_pump*g))) )*A;  % Volumetric
flow rate
WHP     = Rho*g*Q*H;            % Pump water horse power [W]
Eta     = Eta_k1 + Eta_k2*WHP + Eta_k3*PWM + Eta_k4*WHP^2 ...
                + Eta_k5*PWM^2 + Eta_k6*WHP*PWM;          % Efficiency [%]

% Calculate outputs
Mdot    = Rho*Q;                % Mass flow rate [kg/s]
Power   = (Eta/100)*WHP;        % Pump electrical power [W]

% Scale Mdot
Mdot = Mdot*scale;              % kg/s, Get scale value from mask (default =
1)

% Calculate state derivatives
T_dot   = Mdot*(T_in-T)/(V*Rho);    % Pump outlet temperature
P_dot   = (Mdot-Mdot_down)/(V*Rho*(1/E_fluid+D*(1-v/2)/(t*E))); % Pump outlet
pressure

% Mux state derivatives
x_dot  = [T_dot,P_dot];
end
```

A.4 Standard Pipe

A.4.1 Initialization

```
load('Fluids.mat');

fld = 'Water';
Fluid2 = eval(['Fluid.',fld]);

load('Moody.mat');

KL = KL_correction(KL_tube,Num_MFRS,Num_PS,Num_TS,Num_BV,Num_CV);
set_param(gcb,'KL',num2str(KL));
```

A.4.2 Parameters and Callbacks

Figure A.12 shows the standard pipe mask parameters.

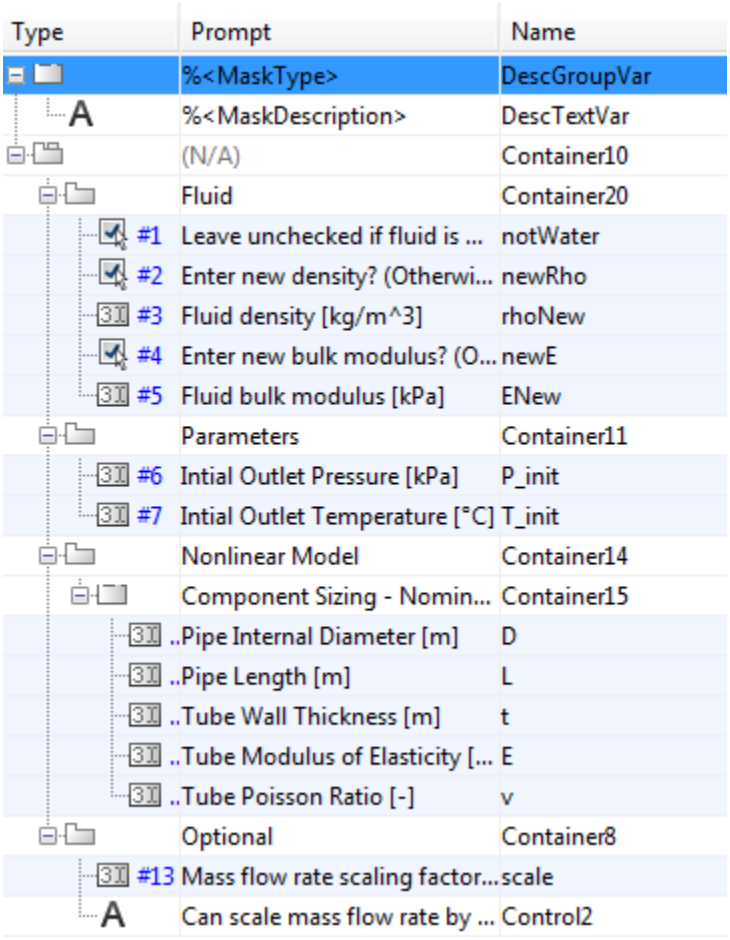

Figure A.12 Standard pipe mask parameters.

The standard pipe has the following callbacks associated with the parameters.

notWater callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'newRho'))
        index_newRho = i;
    elseif (strcmp(names(i),'newE'))
        index_newE = i;
    elseif (strcmp(names(i),'rhoNew'))
        index_rhoNew = i;
    elseif (strcmp(names(i),'ENew'))
        index_ENew = i;
    end
end

notWater = get_param(gcb, 'notWater');

% If user wants to enter a new Rho, provide space
switch get_param(gcb, 'notWater')
    case 'off'
        vis{index_newRho} = 'off';
        vis{index_newE} = 'off';
        vis{index_rhoNew} = 'off';
        vis{index_ENew} = 'off';
    otherwise   %'on'
        vis{index_newRho} = 'on';
        vis{index_newE} = 'on';
end

% set visibility
set_param(gcb,'MaskVisibilities',vis);

fld = 'Water';
Fluid2 = eval(['Fluid.',fld]);
```

newRho callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
```

```
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'rhoNew'))
        index_rhoNew = i;
    end
end

rhoNew = get_param(gcb, 'rhoNew');

% If user wants to enter a new Rho, provide space
switch get_param(gcb, 'newRho')
    case 'off'
        vis{index_rhoNew} = 'off';

    otherwise
        vis{index_rhoNew} = 'on';

end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

newE callback

```
% choose what options to make visible

% collect visibility values of each blue numbered mask parameters (eg for
% entire block)
vis = get_param(gcb,'MaskVisibilities');

% get list of parameter names to know indices regardless of new parameters
% added or order changes
params = get_param(gcb, 'DialogParameters');
names = fieldnames(params);

% find indices of parameters of interest
for i=1:length(names)
    if (strcmp(names(i),'ENew'))
        index_ENew = i;
    end
end

ENew = get_param(gcb, 'ENew');

% If user wants to enter a new Rho, provide space
```

```
switch get_param(gcb, 'newE')
    case 'off'
        vis{index_ENew} = 'off';

    otherwise
        vis{index_ENew} = 'on';

end

% set visibility
set_param(gcb,'MaskVisibilities',vis);
```

A.4.3 Subsystem

A.4.3.1 Structure

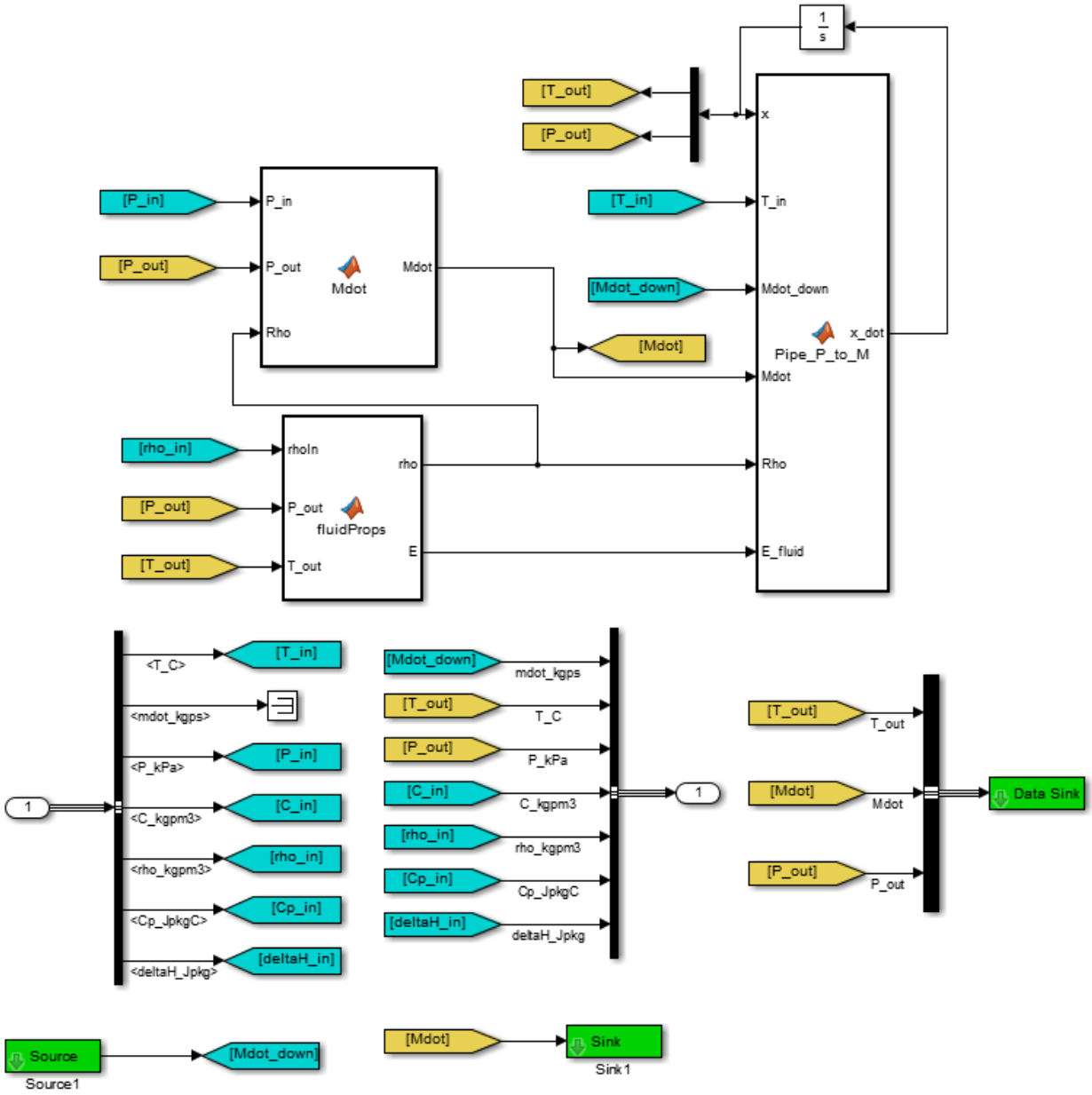

Figure A.13 Standard pipe subsystem structure.

A.4.3.2 Matlab Functions

Mdot Matlab function

```
function Mdot = Mdot(P_in,P_out,Rho,f,D,L,KL,dH)


% Constants and Conversions
g           = 9.81;                 % Gravitational acceleration [m/s^2]
A_cross     = pi*D^2/4;             % Cross sectional area
dP          = (P_in-P_out)*1000;    % Pressure differential [Pa]
dP          = dP + Rho*g*dH;        % Pressure differential corrected for
hieght
um          = sign(dP)*sqrt_Lip(2*abs(dP)/(Rho*(f*L/D+KL)));    % Fluid
velocity [m/s]

% Calculate outputs
Mdot        = um*Rho*A_cross;       % Mass flow rate [kg/s]

end
```

fluidProps Matlab function

```
function [rho, E] = fluidProps(newRho, newE, rhoIn, rhoNew, ENew,...
    P_out, T_out, notWater, Fluid2)
%{
  Calculate E and rho for fluid

  Since there are property tables for water, if fluid is water, then use
  property tables to look up density based on temperature and pressure.

  If it is not water, then probably use incoming fluid density (constant)
  regardless of temprature and pressure.

  Similarly, look up bulk modulus for water or enter own bulk modulus
  (water is used as default, if user doesn't want to enter a bulk modulus).

  **If other property tables are known, then incorporate those.
%}

% if not water (anything else
if (notWater)
   if (newE);
        E = ENew;                   % if user gives E value, use it
   else
        E = Fluid2.Bulk_Mod;                % otherwise, use water E value
   end

   if (newRho)
       rho = rhoNew;
   else
       rho = rhoIn;
```

```
   end

% if Water
else
    % Check that P and T values are within range
    if (P_out < 500)
        P_out = 500;
    elseif (P_out > 20960.8)
        P_out = 20960.8;
    else
    end

    if (T_out < 2)
        T_out = 2;
    elseif (T_out > 500)
        T_out = 500;
    else
    end

    rho = qminterp2(Fluid2.T, Fluid2.P, Fluid2.Rho_pt, T_out, P_out);   %
faster than interp2

    E = Fluid2.Bulk_Mod;
end
```

Pipe_P_to_M function

```
function x_dot  = Pipe_P_to_M(x,T_in,Mdot_down,Mdot,Rho,E_fluid,D,L,t,E,v)

% Constants and Conversions
V       = L*pi*D^2/4;           % Volume

% Parse states
T       = x(1);                 % Temperature at the outlet of pipe
P       = x(2);                 % Pressure at the outlet of pipe

% Calculate state derivatives
T_dot = Mdot*(T_in-T)/(V*Rho);  % Pipe outlet temperature
P_dot = (Mdot-Mdot_down)/(V*Rho*(1/E_fluid+D*(1-v/2)/(t*E)));   % Pipe outlet
pressure

% Mux state derivatives
x_dot  = [T_dot,P_dot];

end
```

A.5 Secondary Pipe

A.5.1 Initialization

```
load('Moody.mat');
```

```
KL = KL_correction(KL_tube,Num_MFRS,Num_PS,Num_TS,Num_BV,Num_CV);
set_param(gcb,'KL',num2str(KL));
```

A.5.2 Parameters and Callbacks

Figure A.14 shows the secondary pipe mask parameters. The secondary pipe has no callbacks.

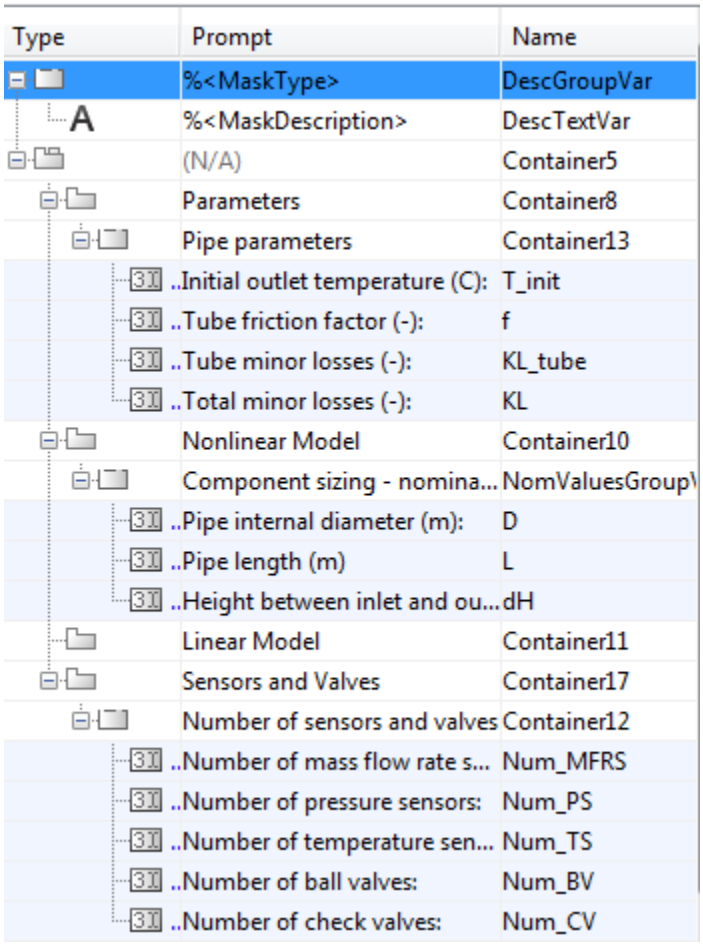

Figure A.14 Secondary pipe mask parameters.

A.5.3 Subsystem

A.5.3.1 Structure

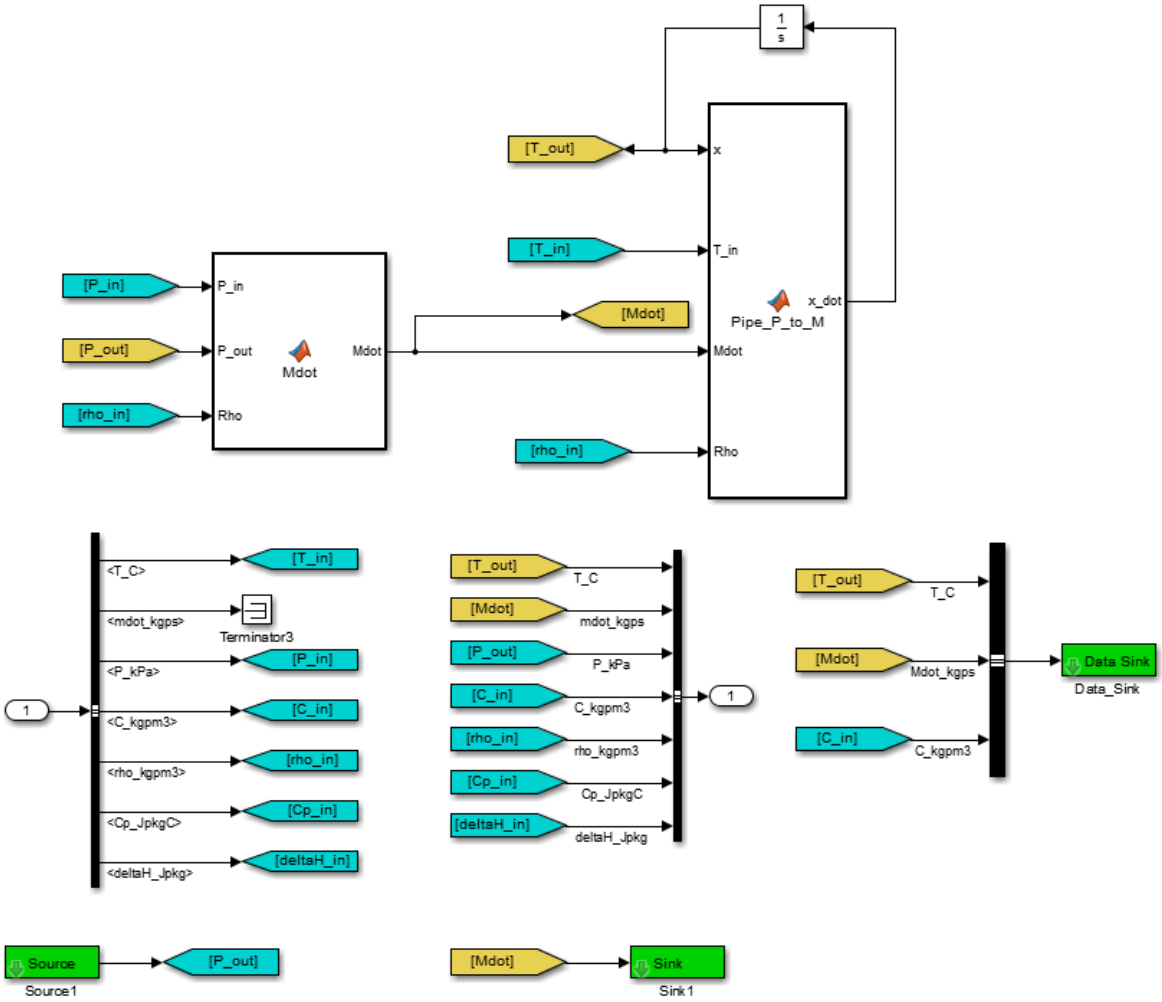

Figure A.15 Secondary pipe subsystem structure.

A.5.3.2 Matlab Functions

Mdot Matlab function

```
function Mdot = Mdot(P_in,P_out,Rho,f,D,L,KL,dH)

% Constants and Conversions
g           = 9.81;                 % Gravitational acceleration [m/s^2]
A_cross     = pi*D^2/4;             % Cross sectional area
dP          = (P_in-P_out)*1000;    % Pressure differential [Pa]
```

```
dP          = dP + Rho*g*dH;        % Pressure differential corrected for
height
um          = sign(dP)*sqrt_Lip(2*abs(dP)/(Rho*(f*L/D+KL)));    % Fluid
velocity [m/s]

% Calculate outputs
Mdot        = um*Rho*A_cross;       % Mass flow rate [kg/s]

end
```

Pipe_P_to_M Matlab function

```
function x_dot   = Pipe_P_to_M(x,T_in,Mdot,Rho,D,L)

% Constants and Conversions
V      = pi*D^2/4*L;            % Volume (m^3)

% Parse states
T      = x(1);                  % Temperature of the tank (C)

% Calculate state derivatives
T_dot  = Mdot*(T_in-T)/(V*Rho);  % Pipe outlet temperature

% Mux state derivatives
x_dot  = T_dot;

end
```

A.6 Valve

A.6.1 Initialization

```
load('Valves.mat')
```

A.6.2 Parameters and Callbacks

Figure A.16 shows the valve mask parameters. The valve has no callbacks.

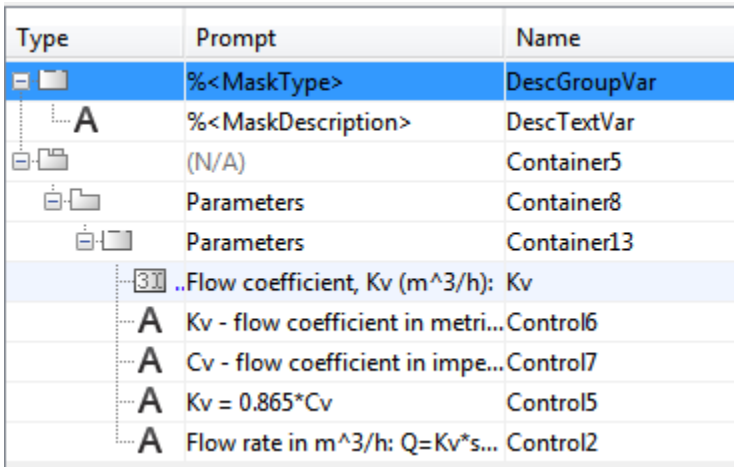

Figure A.16 Valve mask parameters.

A.6.3 Subsystem

A.6.3.1 Structure

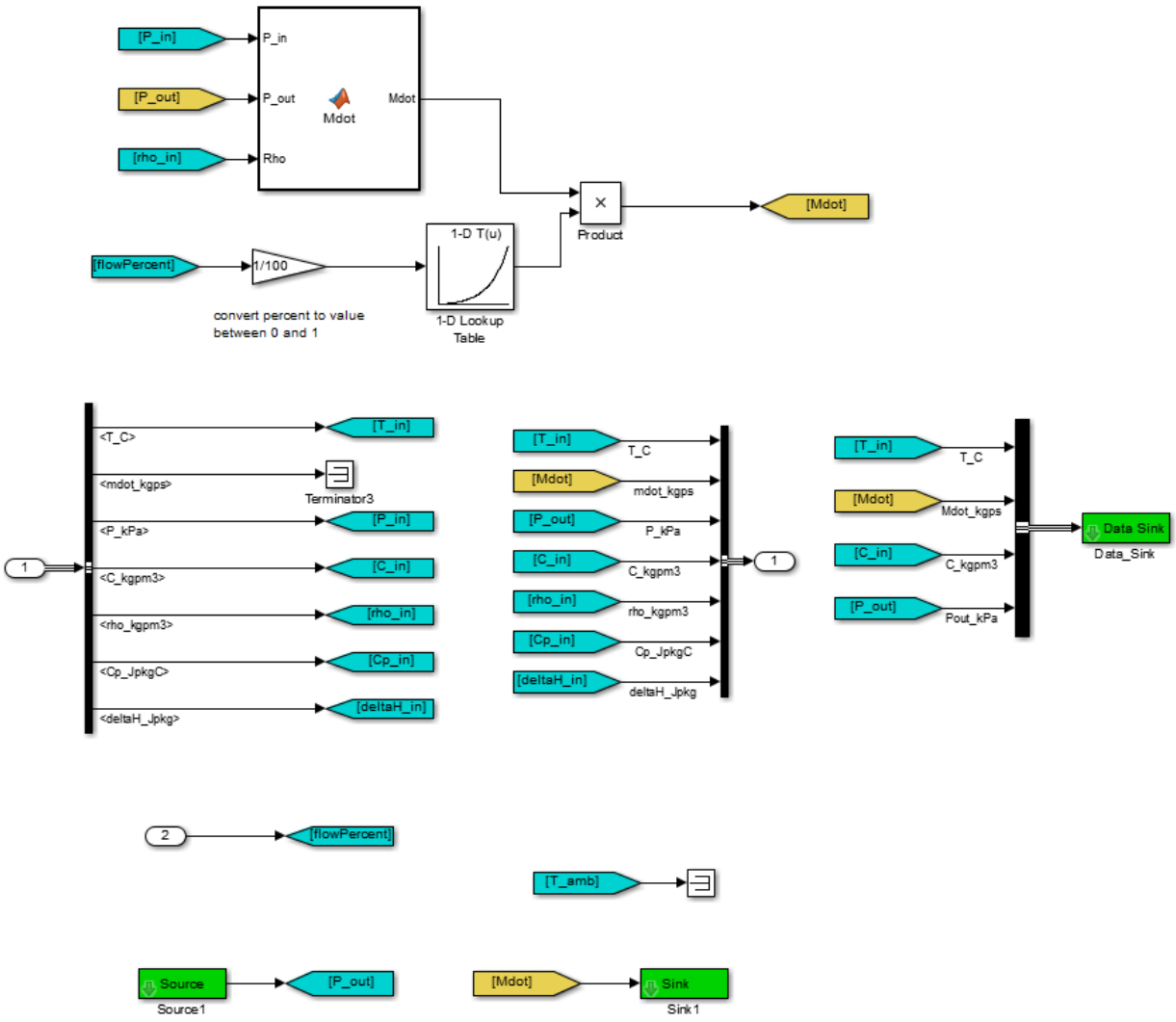

Figure A.17 Valve subsystem structure.

A.6.3.2 Matlab Functions

Mdot Matlab function

```
function Mdot = Mdot(P_in, P_out, Rho, Kv)
%{
Valve component which calculates a mass flow rate based on the inlet and
 outlet pressures. Assumes fluid has the density of water - this can be
```

```
 updated later, if desired.

Reference: http://www.valvias.com/flow-coefficient.php
%}

% constants
Rho_water = 994.1;                  % kg/m^3

% Mdot calculation for a valve
dP = P_in - P_out;                  % kPa, pressure change
dP = dP/100;                        % bar, pressure change
SG = Rho/Rho_water;                 % unitless
Kv = Kv/60;                         % m^3/s (Kv originally has units m^3/h
and expects P in bar=100kPa)
Q = Kv*sqrt_Lip(abs(dP)/SG);        % m^3/s
Mdot = Rho*Q;                       % kg/s

end
```

A.7 Mass Flow Rate Source

A.7.1 Initialization

The mass flow rate source has no initialization code.

A.7.2 Parameters and Callbacks

Figure A.18 shows the mass flow rate source parameters. The mass flow rate source has no callbacks.

Type	Prompt	Name
	%<MaskType>	DescGroupVar
A	%<MaskDescription>	DescTextVar
#1	Mass flow rate (kg/s)	M
#2	Temperature (C)	T
#3	Pressure (kPa)	P
#4	Fluid:	fluidtype
#5	Density (kg/m^3):	rho_mask
#6	Heat capacity (J/kgK):	Cp_mask
#7	Heat of formation (J/kg):	deltaH_mask

Figure A.18 Mass flow rate source mask parameters.

A.7.3 Subsystem

A.7.3.1 Structure

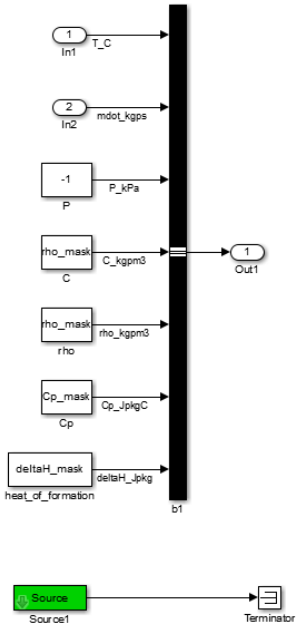

Figure A.19 Mass flow rate source subsystem structure.

A.7.3.2 Matlab Functions

The mass flow rate source subsystem has no Matlab functions.

A.8 Pressure Source

A.8.1 Initialization

The pressure source has no initialization code.

A.8.2 Parameters and Callbacks

Figure A.20 shows the pressure source parameters. The pressure source has no callbacks.

Type	Prompt	Name
	%<MaskType>	DescGroupVar
A	%<MaskDescription>	DescTextVar
#1	Mass flow rate (kg/s)	M
#2	Temperature (C)	T
#3	Pressure (kPa)	P
#4	Fluid:	fluidtype
#5	Density (kg/m^3):	rho_mask
#6	Heat capacity (J/kgK):	Cp_mask
#7	Heat of formation (J/kg):	deltaH_mask

Figure A.20 Pressure source mask parameters.

A.8.3 Subsystem

A.8.3.1 Structure

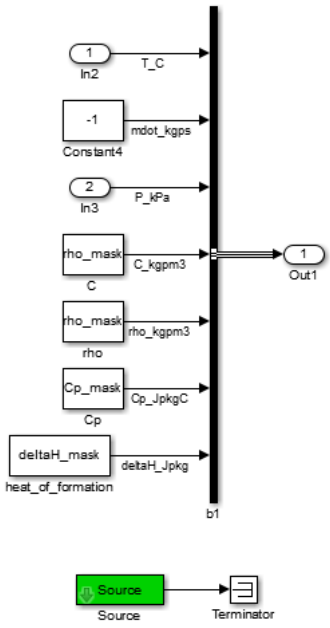

Figure A.21 Pressure source subsystem structure.

A.8.3.2 Matlab Functions

The pressure source subsystem has no Matlab functions.

A.9 Mass Flow Rate Sink

A.9.1 Initialization

The mass flow rate sink has no initialization code.

A.9.2 Parameters and Callbacks

The mass flow rate sink has no parameters or callbacks.

A.9.3 Subsystem

A.9.3.1 Structure

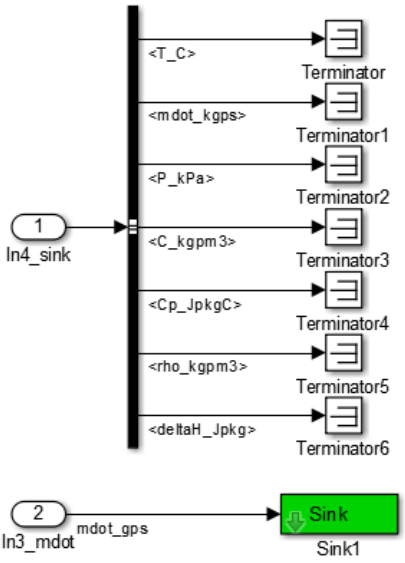

Figure A.22 Mass flow rate sink subsystem structure.

A.9.3.2 Matlab Functions

The mass flow rate sink subsystem has no Matlab functions.

A.10 Pressure Sink

A.10.1 Initialization

The pressure sink has no initialization code.

A.10.2 Parameters and Callbacks

The pressure sink has no parameters or callbacks.

A.10.3 Subsystem

A.10.3.1 Structure

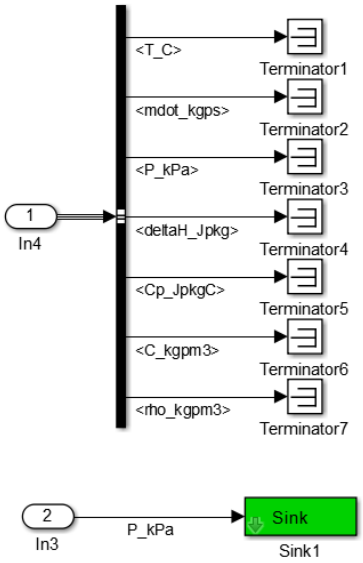

Figure A.23 Pressure sink subsystem structure.

A.10.3.2 Matlab Functions

The pressure sink subsystem has no Matlab functions.

Appendix B

Crystal Violet Reaction Experimental Procedure and Cost

Crystal violet experimental procedures to obtain the experimental results shown in Figure 3.6 and Figure 3.7 are included here.

B.1 Temperature Dependence Procedure

1. Measure 5 mL of 2.7×10^{-5} M crystal violet solution in a 10 mL graduated cylinder and pour the solution into the first 30 mL beaker.
2. Measure 5 mL of 0.1 M sodium hydroxide solution in a 10 mL graduated cylinder and pour the solution into the second 30 mL beaker.
3. Place both 30 mL beakers on the Peltier cooler to heat or cool solutions to the desired temperature. Adjust the temperature by changing the applied voltage of the power supply attached to the Peltier cooler.
4. Once both solutions have reached desired temperature, place the stand, photoresistor, light source, and first beaker in place as shown in Figure 3.3. Keep the applied voltage to the Peltier cooler constant to maintain the same temperature throughout the reaction.
5. Begin acquiring data from the photoresistor with a sampling rate of one sample per second.
6. Pour the second beaker with the sodium hydroxide solution into the first beaker to begin the chemical reaction.
7. The trial is complete when solution is clear.

8. Repeat steps 1-7 for the desired variety of temperatures.

B.2 Dynamic Response Procedure

1. Set up the stand, photoresistor, light source, and beaker as shown in Figure 3.3.
2. Measure 10 mL of 2.7×10^{-5} M crystal violet solution in a 10 mL graduated cylinder and pour into the 30 mL beaker on the Peltier cooler.
3. Measure 10 mL of 0.1 M sodium hydroxide solution in a 10 mL graduated cylinder.
4. Begin acquiring data from the photoresistor with a sampling rate of one sample per second.
5. Pour the sodium hydroxide solution into beaker with crystal violet solution and begin timer.
6. Measure 5 mL of 2.7×10^{-5} M crystal violet solution in a 10 mL graduated cylinder.
7. At 400 seconds, pour the additional 5 mL of cystal violet solution into the beaker.
8. Measure 5 mL of 2.7×10^{-5} M crystal violet solution in a 10 mL graduated cylinder.
9. At 800 seconds, pour the additional 5 mL of cystal violet solution into the beaker.
10. At 1400 seconds, the trial is complete.

B.3 Cost Breakdown

The cost breakdown for gathering the crystal violet reaction data is provided in Table B.1. However, it is important to note a cheaper and less powerful data acquisition system than a myRIO could be used to reduce the experimental setup cost.

Table B.1 Experimental apparatus cost breakdown.

Part	Cost ($)
3D printed stand	4
Beaker, 30 mL	3
Crystal violet solution, 0.024M, 100 mL	8
Sodium hydroxide solution, 1.0M, 500 mL	7
Light source	10
Peltier cooler assembly	35
Power supply (for Peltier cooler)	215
Power supply (for photoresistor)	215
myRIO	249
Total	**746**