Hydraulic Test Cell Power Recovery System

Peter M. Kanda
Western Michigan University, peter.m.kanda@gmail.com

Follow this and additional works at: http://scholarworks.wmich.edu/honors_theses
Part of the Mechanical Engineering Commons

Recommended Citation
Peter Kanda, having been admitted to the Carl and Winifred Lee Honors College in the fall of 2008, successfully completed the Lee Honors College Thesis on April 17, 2012.

The title of the thesis is:

Hydraulic Test Cell Power Recovery System

Dr. Christopher Cho, Mechanical & Aeronautical Engineering

Dr. Jerry Hamelink, Mechanical & Aeronautical Engineering

Mr. Jeff Huffman, CFPE of FEMA Corporation
Hydraulic Test Cell Power
Recovery System

FEMA CORPORATION
ME 4800

Brian Doll
Peter Kanda
Spencer Mick

Faculty Mentor: Dr. Christopher Cho
Industrial Mentor: Jeffrey Huffman

Aeronautical / Mechanical Engineering
Western Michigan University
April 2012
Abstract
Wasted power during testing and development of fluid power components is a major concern due to the negative environmental impact and overall associated energy costs. Dynamometer loading needed to simulate actual component field use results in this power loss. A concept for recouping this loss was developed, modeled, and tested. This energy recovery concept has application for both developmental testing and final production component validation.
# Table of Contents

Abstract ..................................................................................................................................................... 1
Table of Figures ......................................................................................................................................... 4
Table of Tables .......................................................................................................................................... 5
Introduction: FEMA Corporation .............................................................................................................. 6
   Problem Background: ............................................................................................................................ 6
Project Goals ............................................................................................................................................. 6
Benchmarking ........................................................................................................................................... 7
Design Concepts ....................................................................................................................................... 7
   Climate Control Heat Recovery System ............................................................................................ 7
      Background ........................................................................................................................................ 7
      Design Concept/Theory ..................................................................................................................... 8
   Climate Control Recovery System: Pros ........................................................................................... 11
   Climate Control Recovery System: Cons .......................................................................................... 11
Accumulator Series Pressure Circuit ....................................................................................................... 11
Hydraulic Generator Circuit ..................................................................................................................... 13
Circuit Component Definitions ............................................................................................................... 14
   Hydraulic Pumps .............................................................................................................................. 15
   Vane Pumps ..................................................................................................................................... 15
   Gear Pumps ..................................................................................................................................... 16
   Piston Pumps .................................................................................................................................... 16
   Pressure Control Valves .................................................................................................................... 18
   Directional Control Valves ................................................................................................................ 20
Supplemental Components .................................................................................................................... 21
Final Design: Closed-Loop Power Recovery ........................................................................................... 24
   Reference: Four-Square Transmission Test Cell .............................................................................. 24
   Four-Square Transmission Testing .................................................................................................... 27
Final Design: Closed-Loop Power Recovery System ............................................................................... 28
Decision Matrix ....................................................................................................................................... 29
AMESim Model ....................................................................................................................................... 31
AMESim Simulation Components .......................................................................................................... 31
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Building 6 Heating Cost</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>Building 6 Heat Contribution w/100GPM Test Stand</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>Accumulator Series Pressure Circuit</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>Hydraulic Generator Circuit</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>Balanced Vane Pump</td>
<td>16</td>
</tr>
<tr>
<td>6</td>
<td>Variable Displacement Piston Pump</td>
<td>17</td>
</tr>
<tr>
<td>7</td>
<td>Relief Valve</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>Check Valve</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>Directional Control Valve</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>Bourdon Tube Pressure Gauge</td>
<td>22</td>
</tr>
<tr>
<td>11</td>
<td>Bladder Accumulator</td>
<td>23</td>
</tr>
<tr>
<td>12</td>
<td>Gear Pump Diagram</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>Parker Pump Efficiency Plot</td>
<td>25</td>
</tr>
<tr>
<td>14</td>
<td>Closed-Loop Power Recovery</td>
<td>26</td>
</tr>
<tr>
<td>15</td>
<td>Four-square Test Schematic</td>
<td>28</td>
</tr>
<tr>
<td>16</td>
<td>AMESim Circuit</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>Motor Signal Input</td>
<td>32</td>
</tr>
<tr>
<td>18</td>
<td>Variable Speed Motor</td>
<td>32</td>
</tr>
<tr>
<td>19</td>
<td>Rotary Power Sensor</td>
<td>32</td>
</tr>
<tr>
<td>20</td>
<td>Power Conversion - Electric Motor to Pump</td>
<td>33</td>
</tr>
<tr>
<td>21</td>
<td>Hydraulic Pump</td>
<td>33</td>
</tr>
<tr>
<td>22</td>
<td>Pump Parameters</td>
<td>34</td>
</tr>
<tr>
<td>23</td>
<td>Hydraulic Relief Valve</td>
<td>34</td>
</tr>
<tr>
<td>24</td>
<td>Relief Valve Parameters</td>
<td>35</td>
</tr>
<tr>
<td>25</td>
<td>3-port Hydraulic Valve</td>
<td>35</td>
</tr>
<tr>
<td>26</td>
<td>3-way Valve Parameters</td>
<td>36</td>
</tr>
<tr>
<td>27</td>
<td>Tank</td>
<td>36</td>
</tr>
<tr>
<td>28</td>
<td>Hydraulic Motor</td>
<td>37</td>
</tr>
<tr>
<td>29</td>
<td>Hydraulic Motor Parameters</td>
<td>37</td>
</tr>
<tr>
<td>30</td>
<td>Flow Rate vs. Motor Displacement</td>
<td>39</td>
</tr>
<tr>
<td>31</td>
<td>Pressure vs. Motor Capacity</td>
<td>40</td>
</tr>
<tr>
<td>32</td>
<td>Motor Displacement vs. Mechanical Power Input/Output</td>
<td>41</td>
</tr>
<tr>
<td>33</td>
<td>Relief Valve Flow</td>
<td>42</td>
</tr>
<tr>
<td>34</td>
<td>Tank Flow</td>
<td>43</td>
</tr>
<tr>
<td>35</td>
<td>Motor Flow</td>
<td>43</td>
</tr>
<tr>
<td>36</td>
<td>Net Power Usage</td>
<td>44</td>
</tr>
<tr>
<td>37</td>
<td>AMESim Circuit</td>
<td>173</td>
</tr>
<tr>
<td>38</td>
<td>Relief Valve Flow Rate (min: 10.3028 LPM) (max: 81 LPM)</td>
<td>174</td>
</tr>
<tr>
<td>39</td>
<td>Tank Flow (min: -56.6785 LPM)</td>
<td>174</td>
</tr>
</tbody>
</table>
Table of Tables
Table 1: NEMA Electric Motor Efficiency Standards................................................................. 7
Table 2: Building 6 Heat Sources ............................................................................................. 9
Table 3: Decision Matrix for hydraulic test stand design alternatives ..................................... 31
Table 4: Motor and Pump Data ............................................................................................... 38
Table 5: Bill of Materials ......................................................................................................... 45
Introduction: FEMA Corporation

FEMA Corporation is an industry-leading hydraulics company, specializing in the development and manufacture of high-performance electro-hydraulic valves for the agricultural and industrial markets. FEMA Corporation’s engineering lab performs rigorous testing of all FEMA products in order to ensure the robustness of valves delivered to customers.

Problem Background:

Recently developed FEMA valve families control much higher flows, at relatively high pressure, than valves FEMA Corporation has produced over the past 50+ years. These new families which include power-intensive electro-hydraulic steering and satellite navigation valves require much larger and more powerful test stands to simulate field operation of the valves.

The new test stand, which is currently in development, uses a 300+ horsepower pump to recreate the operation of the valve in the customer’s application. In existing test stands, a relief valve is used to control system supply pressure and, in some test systems, simulate a load on a particular line.

While the relief valve is an effective and accurate method of building and controlling pressure, it generates high levels of heat which equates to a loss of energy to the atmosphere. In lower-output test stands, this loss is relatively insignificant. As the test stand output increases, though, this loss becomes more noticeable. Preliminary calculations indicate the power lost through the EF line (described in detail in Requirements and Specifications) can be more than 150 horsepower in hydraulic kinetic energy. Putting that loss into perspective, there are at least five test stands in the FEMA engineering lab that run on less than 30 horsepower.

The power loss across the relief valves of high-output test stands is no longer negligible, and it is in FEMA Corporation’s best interest to find an effective method of recouping this lost power. It is also desirable to design a system that is applicable to present and future test cells in the engineering lab.

Project Goals

Develop alternate way to control system supply pressure and loading that does not generate high levels of heat and loss. With high amounts of fluid being forced through a small orifice, friction causes everything to heat up. Most of the energy in the current test stand is lost due to a substantial increase in heat.

Implement method to recover lost energy and turn into useful energy. By creating a system that could recoup some of the energy used and putting it back into the test stand would be very beneficial.

Design a system that is adaptable to current and future test stands. Making an energy system that could be used not only on one test stand, but different set ups would be useful and time-efficient for technicians and engineers using the test equipment.
Benchmarking

Design Concepts

Climate Control Heat Recovery System

Background
The building in which the new test stand is installed housing many other hydraulic systems in both the engineering lab and multiple production assembly cells, which is heated mainly by these test stands. The machinery involved accounts for much of the heat used to control the temperature in this building during the colder months. In warmer months, excess heat is either vented outside or simply negated with the HVAC system (which uses additional energy in itself). This is a separate issue outside the scope of the project, but does show that most of the energy used for product testing is in some sense recouped for heating the building during winter months.

Table 1 below shows the NEMA Design B (insert reference) minimum required efficiencies for electric motors of given output ranges (output in HP). Although power lost through motor inefficiency cannot be recouped, it is necessary to calculate operating cost and also will put into perspective the overall system efficiency.

<table>
<thead>
<tr>
<th>Power (hp)</th>
<th>Minimum Nominal Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - 4</td>
<td>78.8</td>
</tr>
<tr>
<td>5 - 9</td>
<td>84</td>
</tr>
<tr>
<td>10 - 19</td>
<td>85.5</td>
</tr>
<tr>
<td>20 - 49</td>
<td>88.5</td>
</tr>
<tr>
<td>50 - 99</td>
<td>90.2</td>
</tr>
<tr>
<td>100 - 124</td>
<td>91.7</td>
</tr>
<tr>
<td>&gt; 125</td>
<td>92.4</td>
</tr>
</tbody>
</table>

Using an average of 90% efficiency for each motor in operation in the building, the heat contributed to the building from each source was calculated. The heat lost was converted into BTU/hr, and multiplied by 10 to account for the 10-hour active work day. This number was reduced
by 25%, a conservative estimate for breaks, down-time, and other work stoppages. Figure 1 shows the division of heat sources currently used to heat Building 6. The furnace values were averaged from electrical bills. The results show that the furnace accounts for less than 1% of the total heating source for the building.

![Figure 1: Building 6 Heating Cost](image)

**Design Concept/Theory**

Table 2 below shows the heat (in BTU) lost from each Building 6 test stand, as well as the contribution from the furnace and the total heat generated in the building. The last row is the calculated heat generated from losses on the new 100GPM test stand. An efficiency of 95% was used for the 100GPM test stand calculations, based on the NEMA standards outlined in Table 1. The projected heat dissipated from the new test stand is equal to 78% of the sum of the current heat sources in the building. The theoretical division of heat sources is shown in Figure 2.
### Table 2: Building 6 Heat Sources

<table>
<thead>
<tr>
<th>Heat Source</th>
<th>HP</th>
<th>BTU Losses w/75% On Time</th>
<th>BTU Losses/Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ernie</td>
<td>75</td>
<td>143,118.56</td>
<td>4,293,556.88</td>
</tr>
<tr>
<td>High Flow</td>
<td>40</td>
<td>76,329.90</td>
<td>2,289,897.00</td>
</tr>
<tr>
<td>Fatigue</td>
<td>10</td>
<td>19,082.48</td>
<td>572,474.25</td>
</tr>
<tr>
<td>Durability</td>
<td>10</td>
<td>19,082.48</td>
<td>572,474.25</td>
</tr>
<tr>
<td>10K</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>ELMO</td>
<td>25</td>
<td>47,706.19</td>
<td>1,431,185.63</td>
</tr>
<tr>
<td>Analog 1</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>Analog 2</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>Analog 3</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>Analog 4</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>DDV</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>PPC 1</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>PPC 2</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>Analog Oven</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>PPC Oven 1</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>PPC Oven 2</td>
<td>5</td>
<td>9,541.24</td>
<td>286,237.13</td>
</tr>
<tr>
<td>Furnace</td>
<td>350</td>
<td>2,456.58</td>
<td>73,697.50</td>
</tr>
<tr>
<td>Building 6 Current</td>
<td>521,905.36</td>
<td>12,728,002.82</td>
<td></td>
</tr>
<tr>
<td>100GPM</td>
<td>350</td>
<td>333,943.31</td>
<td>10,018,299.38</td>
</tr>
</tbody>
</table>
Using the BTU/hr output calculated in the table above, 0.75% of the 100GPM test stand’s heat output will replace the heat output by the furnace, eliminating the need for the furnace altogether. However, there are several issues with this approach.

The high levels of heat loss from the test stand far exceed the furnace output. This means a ducting system must be installed to control the flow of heat into the building when demanded and route the heat out of the building when it is not necessary. This is therefore an inefficient method of recovering the heat. Converting back into horsepower, only 0.04HP is recovered on average throughout the course of the month. All heat not used is routed outside the building, and therefore wasted as a source of energy. Furthermore, the furnace is only used for approximately 4 months out of the year. For the remaining 8 months, no energy would be recovered at all in this manner.

The start-up capital and time of investment recovery for this design is also an issue. Employee safety is the first priority in installing any system into the building, and many precautions must be taken to prevent the heat recovery system from posing potential harm to employees. Filters must be installed into the ductwork to prevent oil vapor and other particles from being circulated into the production floor. Smoke detectors are necessary to shut gates in the ductwork in the event of a fire or motor failure. After these elements are added to the cost of ducting and installation, the total investment is estimated at $10,000. If another medium-scale test stand was added in the future, the heat from that test stand could produce a similar effect on the heating bill (in the form of a passive heating system, as is used with the other test stands) without the costly investment.

The monthly cost of furnace operation is divided into an average of $536.09/month in gas and $317.69/month in electric during the winter months. Keeping in mind the fact that this recovery only occurs during 4 months of the year, the estimated time needed to recoup the investment cost is 2.93 years. Considering the high startup cost, low efficiency, and length of time to recoup investment, other methods of recovering waste energy must be considered and compared to the Climate Control
Heat Recovery System. It was ultimately determined that this is neither the most effective nor the most useful approach for power recovery.

**Climate Control Recovery System: Pros**
- Reduce heating bill
- Non-invasive to test procedure
- Remove furnace – less maintenance/replacement cost
- Versatile – can be used regardless of test procedure

**Climate Control Recovery System: Cons**
- High capital investment
- Length of time to recoup investment
- Low overall efficiency
- Only useful during colder months
- Potential safety hazards
- Limits impact of future test stands/equipment on lowering costs

**Accumulator Series Pressure Circuit**

The construction of this particular hydraulic system is designed to harness the energy storage capabilities of a hydraulic accumulator. The assembly is very straightforward. This circuit starts with the given pump-motor of all of the design concepts and connects it to the customary directional control valve, much like the previous system that this project was challenged on improving. The circuit then differentiates itself from the standard setup by installing an accumulator following the first directional control valve and preceding another directional control valve. In this configuration, one could have an indefinite amount of valves and accumulators, but for this concept there would be a total of four directional control valves with three accumulators in between each valve. The use of four directional control valves establishes a reasonable amount of components for a given technician to observe while utilizing the theory of the concept. The circuit would also include check valves preceding and following the accumulators, to ensure that flow would go in the appropriate direction.
The theory behind this particular setup lies in the effectiveness of the accumulators. Excess pressure and flow will be supplied from the pump to the first directional control valve, which would be captured by the first accumulator. Once the first valve is finished operating, the accumulator would discharge into the second directional control valve for operation. The process repeats itself until the last valve has operated and the flow goes back to the reservoir, where the pump can redraw fluid for the next cycle.

There are obvious benefits to this kind of system. Accumulators and check valves are simple components in a hydraulic system; little to no research needs to be made to understand the behavior of either mechanism, and both are relatively cheap. Prices can vary from the inexpensive piston-operated accumulators starting around $250 all the way up to high performance bladder accumulators for over $1000. The purposes of this project will not require an extensive accumulator, thus the cost should be significantly less when compared to other design concepts in this report. It is also a simple system to construct because the components are connected in series and it is very easy to visualize the path of the fluid. The final benefit stems from the high efficiencies associated with accumulators, usually on the order of greater than 95% efficiency.

The drawbacks of this particular setup lay in the coordination of accumulator discharges. While pressure is building in the first accumulator, the second directional control valve cannot operate, nor can the other components further down the system. Later components can only function after the preceding component has finished operating. Obviously there needs to be some sort of synchronization of the accumulators and/or the directional control valves to make sure the system runs as efficiently as possible. Because we would most likely be working in intervals equating to fractions of a second, this simple system requires significant upgrades in instrumentation and capital investment. These upgrades must improve the lab testing software to ensure the accurate capture of thousands of intervals of test data. Improvements would have to be made in the functionality of the accumulators and the directional control valves, where an automated electronic
circuit could coordinate the flow of fluid in a timely matter. The authors of this report and the industrial mentor for this project decided against this circuit for the aforementioned reason.

**Hydraulic Generator Circuit**

The goal of this specific hydraulic system is to take the excess hydraulic power produced from the system pump and convert it into usable electricity. Like the circuit discussed previously, the hydraulic generator circuit is simple. Starting with the given pump-motor of all the design concepts and a directional control valve, the flow then enters an accumulator to build pressure and store the excess energy coming from the pump. When the accumulator discharges, the flow will exit smoothly into a hydraulic motor. Because of the steady power input from the accumulator, there is a smooth power output from the hydraulic motor that transfers energy into a generator. Essentially the hydraulic motor is taking the excess energy of the pump and is converting it into mechanical energy that operates the generator. The electrical energy produced from the generator can then be used on other hydraulic test stands or can be directly introduced into FEMA’s electrical grid. The end goal would be to save money on the energy bill since some of it will be recycled from this test stand into other test stands or the facility as a whole.

![Figure 4: Hydraulic Generator Circuit](image)

1. Variable Displacement Pump
2. Check Valves
3. Customer Direction Control Valve
4. Accumulator
5. Relief Valve
6. Electric Motor
7. Pressure Regulator
8. Hydraulic Motor
There are clear advantages and benefits to trying to implement this particular setup. Like other circuits mentioned in this report, the construction is very simple. Each component is connected in series with little additions needed, such as relief or check valves, to guide the flow of fluid. The obvious advantage with this setup is the versatility that comes from converting the excess fluid power of the system pump into usable electric power. The energy bill will improve whether the electricity goes into neighboring test stands or helps power the FEMA facility entirely.

Cost is the one significant disadvantage when considering the feasibility of this system. Relatively speaking, hydraulic motors and generators are more expensive features to add to a hydraulic test stand. The cost does not stop at just the motor and generator though, because more money would have to be invested in the shuttling of electrical power from the generating circuit. This process becomes exponentially more expensive when one chooses to enter that power into the FEMA electrical grid because of the necessity of a synchronous inverter. A synchronous inverter changes the DC current from a source into AC current that can be fed into the utility grid. The models that would apply to this system cost thousands of dollars, thus rendering the purchase of a synchronous inverter into a long-term investment. Such an investment is not impossible to calculate, but the large cost of a synchronous inverter requires necessary consideration to ensure capital invested makes sense.

**Circuit Component Definitions**

Hydraulic fluid power is a very powerful phenomenon when trying to apply mechanical power to a different location. The flexibility, convenience, simplicity, and efficiency associated with a given hydraulic circuit makes hydraulic fluid power a widely utilized subsystem within modern machinery today. One would be hard-pressed to find a manufacturing or industrial mechanical capital that did not employ some sort of hydraulic system within its mechanical capital. The project contained within this report is only one example of the thousands of hydraulic systems that could be contained within a hydraulic testing facility. The circuit we have chosen has many features and components that are common among all hydraulic circuits. Some of these common features and components include pumps, relief valves, check valves, directional valves, and necessary instrumentation. Other components that are related to this particular circuit are accumulators and hydraulic motors. It is important to understand the individual components that go into a prospective hydraulic circuit so that one can see the benefits and drawbacks to a particular configuration.

The first characteristic to identify with all of the potential circuits this report highlights is that each circuit can be categorized as an open loop circuit. By definition, an open loop circuit features a pump that draws fluid from a large reservoir that is used for the system. Pumps in an open loop setting can only pump fluid in one direction, and the inlet port is usually larger than the outlet port. The advantage of an open-loop design is that a single pump can be used to drive many different
actuator functions simultaneously\(^1\). As mentioned before, all of these circuits involve the monitoring of components being tested for a customer, so the flexibility of an open-loop circuit makes the most sense for the designs of this project.

**Hydraulic Pumps**

Several features of a hydraulic circuit have already been mentioned in the previous text without a real definition, one of the most prominent being the usage of a pump. Pumps are very important to a hydraulic system, in fact almost all hydraulic systems feature one. In the simplest sense, a hydraulic pump raises the energy level of the fluid, from the low energy supply in the reservoir to the closed hydraulic system where high energy fluid is used to accomplish work\(^2\). In reality, there are many processes that a pump can utilize in order to raise the energy level of a fluid, which in turn defines the type of pump. These types can be broken down into vane, gear, and piston pumps with fixed or variable displacement capabilities. Knowing the variety of pumps available allows for proper pump selection in the construction of a hydraulic circuit.

**Vane Pumps**

Vane pumps generate pumping action by causing vanes to track along a ring. The rotor of a van pump houses the vanes and it is attached to a shaft which is connected to a prime mover\(^3\). These kinds of pumps can have a balanced or unbalanced design. In a balanced design pump, there are opposing pairs of internal inlet and outlet ports which distribute the thrust force evenly around the shaft\(^1\). Because of the unbalanced thrust force within an unbalanced vane pump, bear life becomes shortened, thus making almost all modern vane pumps of the balanced design.

\(^1\) George Totten, Handbook of Hydraulic Fluid Technology.
\(^3\) Parker Training Department, Motion & Control Group, Fluid Power Basics.
Gear Pumps

The gear pump is often regarded as the simplest and most robust kind of pump, perhaps owing to having only two moving parts. Gear pumps generate a pumping action by meshing and un-meshing gears. A typical gear pump contains a housing with inlet and outlet ports, and a drive and driven gear that make up the pumping mechanism. The specifications of the gears such as number of teeth, the pitch circle diameter, and the width of the gears are the dominant parameters that control the displacement of fluid. When the teeth un-mesh at the inlet, fluid enters the housing. Fluid is then trapped between the gear teeth and the housing which is then carried to the other side, where the fluid exits as the teeth mesh together.

Piston Pumps

The last and most complicated type of pump discussed in this report is the piston pump. These kinds of pumps generate pumping action by causing pistons to reciprocate within a piston bore, not all that different from an automobile engine. Practical applications of this method employ multiple cylinders and pistons to smooth out fluid delivery. Piston pumps are manufactured in the axial, bent axis, and radial configurations with the option of being variable- or fixed-displacement. Because of the sheer complexity of each individual arrangement, this report will discuss the axial variable displacement model of piston pump since this will be the pump featured in every hydraulic circuit. Variable displacement piston pumps incorporate the pistons within the cylinder block with piston shoes and a swash plate. The drive shaft rotates the cylinder block with the pistons, which are held against the swash plate by springs and a retainer plate. For the pump to produce a flow, the swash plate must be at some angle relative to the centerline of the shaft, which is also the center of
rotation for the pistons\textsuperscript{1}. The greater the swash plate angle, the larger the piston stroke and the greater the displacement of the pump.

It is widely known that there are a variety of pumps available, but the real question lies in how to properly choose a pump. The sizing of the system pump starts with the load since a hydraulic system’s primary objective is to move a load. Pumps generate a flow and pressure results from the confines of the system and the requirements to maintain a load. Therefore, the first parameter when sizing a pump is to determine the required flow rate with the second most likely being the pressure capability\textsuperscript{1}. Piston pumps are operated at the highest pressure of all the pumps normally found in hydraulic applications. Variable displacement piston pumps allow for the incorporation of various valve mechanisms that will alter the performance of the pump, such as a pressure compensator which alters the displacement of the pump to limit the outlet pressure to a preselected value.

\textbf{Figure 6: Variable Displacement Piston Pump}

This project’s primary focus lies within energy consumption and like any mechanical device, pumps are not 100\% efficient. Pump efficiency can be broken down into to two categories: the volumetric efficiency, which relates actual to theoretical volume delivered, and the power efficiency, the output hydraulic power versus the input mechanical or electrical power. Typical pump efficiencies range around 90\% for gear pumps to about 98\% for high quality piston pumps\textsuperscript{4}. As mentioned before, the pump of focus within this project is a variable displacement piston pump which should allow for greater recovery of power losses when compared to that of a gear or vane pump.

\begin{center}
\textsuperscript{4} Andrew Parr, Hydraulics and Pneumatics: A technician’s and engineer’s guide.
\end{center}
Pressure Control Valves

Now that a pump has been introduced to a hypothetical system, there must be some way to control the fluid that the pump is using throughout the system. This can be done with pressure control valves. A valve is a mechanical device consisting of a body and an internal moving part which connects and disconnects passages within a body. The scope of this report and the hydraulic system contained within it does not require the definitions of the variety of pressure control valves that exist. Instead, this report will cover the basic functions of three fundamental pressure control valves: needle, relief, and check valves.

The simplest method for uncompensated flow control is to use a needle valve with a fixed-area orifice. The needle within the valve can be adjusted to increase or decrease the effective standard orifice size and create a change in fluid flow and pressure. The pressure drop across the effective orifice is proportional to the flow going through an orifice. Therefore, if the pressure differential increases, flow will also increase.

Relief valves are the simplest and most common type of pressure control valves. They can be placed in a variety of positions within a circuit, but almost always there is a relief valve near the pump outlet to protect the pump as well as provide the system and its components protection against overloads. They also limit the output force exerted by cylinders and rotary motors. The two major kinds of relief valves are direct-acting and pilot operated. For a direct acting model, flow enters through an orifice, and when the pressure builds at the inlet to a value such that the pressure times the exposed area exceeds that of the spring setting, the valve will begin to pass hydraulic fluid through the outlet. Other relief valve models may have a preset fluid charge with an accompanying piston as the resist force for the building of pressure. The pressure at which the valve first begins to open is called the cracking pressure. The pilot-operated valve increases pressure sensitivity and reduces the pressure override (the difference between cracking pressure and full-flow pressure) normally found in direct-acting relief valves.
The third kind of pressure control valve discussed in this report is a check valve, which is primarily used to control direction of fluid flow. This valve normally consists of a valve body with an inlet, outlet, and a movable member which is biased by spring pressure. A check valve requires a small cracking pressure and behaves like a low pressure relief valve would. More specifically, a check valve will consist of a seat, poppet, and a spring. The valve will remain closed until sufficient pressure at its inlet will create enough force to overcome the spring force, permitting flow around the poppet and through the valve. If flow tries to enter through the outlet, the poppet is pushed on its seat and flow is blocked. Hence, flow can only go in one direction with a check valve. This concept is vital to the circuit contained in this project because energy recuperation will require as little energy loss due to fluid flow as possible, so the importance of check valves cannot be understated.
Directional Control Valves

The next logical step in the understanding of hydraulic components is to understand the mechanisms that control the direction of flow through a hydraulic system. In theory, the check valve mentioned before functions as a directional control valve because flow is restricted in one direction. However, valves with multiple orifices are needed to execute the actions required of a hydraulic system.

Three terms in particular are used in the description of directional control valves: position, way, and port. Position refers to the placement within the valve body of a shifting mechanism or element, such as a sliding spool in a spool valve. Way refers to the flow path through the valve, including reverse flow. One-way, three-way, and four-way valves are common. A simple check valve, as mentioned above, is a one-way valve. Two-way valves allow fluid to flow in both directions in one fluid passage through the valve. The hydraulic circuits developed for this project utilize a four-way valve, which are reversing valves allowing fluid to flow in both directions in two passages to operate a cylinder, motor, or other device in the forward and reverse direction. The four ports of a four-way valve are typically identified as pump (P), reservoir (T), and the two working ports (A and B). Sliding spools and rotating shear plates are the most common internal valve elements used to accomplish the shifting. The raised areas called “lands” block or open ports to give the required operations. In the case of test circuit being modeled for this project, the four-way valve operates a steering valve for a particular customer product.
Supplemental Components

The features that have been discussed thus far can be almost described as essential when it comes to the construction of a hydraulic circuit. It is hard to find a hydraulic system that does not contain a pump with pressure and directional control valves. The following features may not be deemed as “essential” but they certainly are integral to the discussion of hydraulics, especially in consideration with the hydraulic circuits discussed in this report.

One of the more obvious component choices that should be discussed involves the components dealing with instrumentation. While a hydraulic system may be able to run with the above components, one would not be able to properly observe the circuit without the proper equipment. The five parameters that are commonly monitored by a technician or engineer are system pressure, temperature, flow rate, speed, and fluid level. While all of these parameters are important, the purposes of this project only require strict observation of the pressure and flow rate since they are the primary values related to energy. There are a variety of pressure gauges, but the most common for all ranges is the bourdon pressure gauge because of its simplicity, cost, and robust construction. For the purposes of this project, the pressure device of choice is a piezoelectric pressure gauge. A pressure sensor within the device acts as a transducer, converting fluid pressure into an electronic signal read by a data acquisition system. Fluid flow is measured to determine the volume flow rate in the system from the pump or through an actuator. Fluid flow devices can measure flow with a simple sight indicator or with a small turbine that displays the flow on a digital readout in gallons per minute or another convenient scale.
Another major component that is extremely important in regards to the circuits described in this report is called an accumulator, as shown in Figure 4. The goal of an accumulator in a hydraulic system is to store or provide fluid at a pressure to diminish quick pressure spikes or to reach a short-duration high flow requirement. They can use ballast weights, springs, or gas pressure to generate the pre-charge force against the fluid that is stored under pressure for use in the system. Gas-charged accumulators use pistons, diaphragms, bladders, or no medium whatsoever to split the two fluids. In many ways, an accumulator functions the same as a capacitor in an electronic power supply. Once the hydraulic pressure exceeds the pre-charge pressure exerted by the gas, the gas will compress, allowing hydraulic fluid to enter the accumulator. In terms of the circuits mentioned in this report, an accumulator is essential when trying to recover the power lost in the system. Smoother fluid transfer allows for efficient power transfer and the maintaining of power, so the benefits of including an accumulator are very attractive for the scope of this project.
The last major feature discussed in this report is called a hydraulic motor. In the most basic sense, a hydraulic motor performs the opposite task of a hydraulic pump. A hydraulic motor converts the high energy level of the fluid to a usable form of mechanical energy by applying the fluid to the movable internal mechanism of an actuator or drive shaft. Therefore the movement of an internal mechanism lowers the energy level of the fluid and accomplishes the objective required of the hydraulic system, and returns the fluid to the reservoir at a low pressure. Rotary actuators can be classified the same way pumps are by being gear, vane, or piston motors with fixed or variable displacement. Cost and efficiency are typically lower with gear motors and higher for piston motors that represent the highest available standard in the industry with respect to precision engineering and machining. For the purposes of this project, a hydraulic motor will be the primary source for energy conversion. Possible ways to output the mechanical energy from the hydraulic motor are to channel it to a generator to create usable electricity, or the mechanical energy can be immediately recycled to reduce the energy consumed by the system pump. The driving force behind the choice of energy output is energy efficiency. Converting energy into different forms, such as mechanical or electrical, results in efficiency losses, therefore the goal of this project is to limit the amount of energy conversions so as to maximize the amount of energy recouped for the system.
Final Design: Closed-Loop Power Recovery

Reference: Four-Square Transmission Test Cell

In any energy-recovery system, component efficiency plays a vital role in determining the effectiveness of the design. The total available energy is reduced through efficiency losses each time energy changes form. The flow chart below outlines the energy state changes through the hydraulic generator circuit.

In order to calculate the available energy at the end of the loop, it is necessary to account for efficiency losses at each stage of the system. As shown in Table 1, minimum motor efficiencies range from 78.8 to 92.4 percent, depending on the size of the motor. For this approximation, it can be assumed that the motor is a high-output motor in good working condition, and an efficiency of at least 90% is expected.

When converting rotational energy to hydraulic kinetic energy necessary for hydraulic operation, there are multiple factors that affect system efficiency. The electric motor must overcome friction of the pump shaft and gears. There is also a case drain associated with hydraulic pumps. This occurs when the hydraulic fluid leaks through the gap between the gear teeth and pump housing.
A pump in good condition should operate around 90% overall efficiency at ideal pressure and flow. The figure below (Parker reference catalog) shows that at 1800 RPM and 150 bar outlet pressure, the pump used for testing (see “experimental testing and analysis”) will operate at approximately 90% overall efficiency.

Through the hydraulic loop, the plumbing and hydraulic components, such as orifices and valves, will create a friction loss. Since each test is different, it is not possible to determine the contribution to efficiency loss from system friction and will therefore be neglected by assuming constant pipe lengths with the same hydraulic components during testing and mathematical simulation.
As the hydraulic fluid enters the hydraulic motor, it experiences the same friction and case drain losses seen in the pump. In this case, an efficiency of 90% will be assumed, although each motor will vary based on the same parameters shown in the efficiency plot above.

The hydraulic motor will turn a generator that will feed back into the electrical grid. The generator is where the most severe efficiency loss takes place: most generators’ efficiencies fall into the 60-80% range. The grid-tie inverter used to transfer the generator’s electricity back into the power grid has an efficiency of approximately 95%.

Figure 14: Closed-Loop Power Recovery

1. Variable Displacement Pump
2. Check Valves
3. Customer Direction Control Valve
4. Flow Meter
5. Pressure Gauge
6. Relief Valve
7. Electric Motor
8. Generator
9. Tachometer
The overall system efficiency is obtained by factoring in each individual component’s efficiency. The breakdown is given below:

- Electric Motor: 90%
- Pump: 90%
- Hydraulic Motor: 90%
- Generator: 70%
- Grid-tie Inverter: 95%
- System: 48.5%

Under ideal operating conditions, only 48.5% of the input power would be available for regeneration at the end of the loop. This does not account for friction losses in the system, nor does it account for the cycling of the valves. The test process itself will reduce efficiency in multiple ways. First, the test valves change flow direction, meaning the fluid flow is only available when the valves are open. If the on/off times for the valves are equal, this means only half of the system flow is available for regeneration (theoretically reducing available regenerated power to 24.25%). Also, the cycled valves cause pressure spikes downstream of the valves, making an accumulator necessary to level out the spikes and keep a continuous pressure and flow supply to the hydraulic motor. The accumulator has an average efficiency of approximately 95%. Without taking into account any other factors, this drops the maximum recovered power to 23% of the system input.

Removing the generator (and therefore the grid-tie inverter) increases the maximum recovered power to 35%. In order to remove the generator from the system, the power generated by the hydraulic motor must be tied directly back into the test loop. Driving the pump with the hydraulic motor should decrease the amount of power required from the electric motor, therefore reducing current draw and lowering operating cost. A similar procedure is followed in the transmission testing industry using a method called “four-square testing”.

**Four-Square Transmission Testing**

Research was conducted to determine if energy-recovery loops were used elsewhere in industry. In the transmission test and development industry, some companies will use a “four-square” testing method. The test method will use test two transmissions at once. Once the required torque is applied, the energy will stay within the closed-loop system, and the motor is used only to make up for friction and efficiency losses. By keeping the energy within the loop (instead of, for example, spinning a disconnected shaft) a high-powered test can be performed with relatively little energy consumption by the drive motor. A diagram of a four-square transmission test is shown below.
Although the four-square test stand is a highly efficient method of recouping durability test power loss, it does not correlate well to the problem presented by the project. In transmission testing, all energy stays in one form (rotational). As previously outlined, the energy in a hydraulic durability test changes states several times throughout the test process. In addition, in hydraulic testing the test pieces are often cycled, or turned on and off at a specified frequency (often around 4 Hz). Loading and unloading mechanical equipment at this rate can cause excessive wear to moving components, increasing maintenance frequency intervals and associated costs.

Inspired by the innovation of the four-square test method, the final design will attempt to apply a similar principle to a hydraulic test cell. The challenges of applying this concept to a fluid power system are outlined in the “Final Design” section below.

**Final Design: Closed-Loop Power Recovery System**

After reviewing the benefits and drawbacks of the electric-generator power recovery system, it was determined that removing the generator from the loop would allow for a more compact and efficient system. The generator consumes valuable space in the test chamber and also degrades the efficiency of the system as a whole, as outlined in the previous section. In order to remove the generator, the hydraulic motor will instead be coupled directly to the pump. The torque generated by the hydraulic motor will be applied directly to the pump, therefore reducing the amount of torque
required from the electric motor. This reduces the electric current drawn by the motor, lowering the operating cost of the test cell.

In practice, the pressure and flow applied to the hydraulic motor is dependent on the nature of the test being conducted and will therefore vary from test to test as well as within the duration of each trial. The pressure and flow must be leveled using a hydraulic accumulator upstream of the hydraulic motor and downstream of the test piece, as shown in (figure of hydraulic circuit). The function of a hydraulic accumulator is described in (named report section). This will reduce the effect of pressure spikes introduced by test cycling and allow for a consistent power output from the hydraulic motor.

Since the hydraulic motor is coupled directly to the pump and electric motor, it will run as a pump when there is no flow at the inlet driving the gears. Running a pump, in this case the hydraulic motor, dry, without a fluid running through it, can cause damage to the pump and dramatically reduce component life. In order to prevent such an occurrence, the hydraulic motor’s inlet will be plumbed into the reservoir using a low-pressure check valve as shown in Figure 4. When there is no pressure in the inlet line, the check valve will open, allowing the hydraulic motor to act as a pump. This will prevent the motor from being damaged by running dry. When the motor is acting as a pump, the current drawn by the electric motor is expected to increase, since the electric motor is responsible for powering two pumps. However, when pressure is sensed by the check valve in the inlet line, it will close off the path from the reservoir to the hydraulic motor. When the flow is supplied to the hydraulic motor from the outlet of the test cell, the motor will convert the kinetic energy to rotational energy and apply it directly to the pump, reducing the current draw of the electric motor.

**Decision Matrix**

To better understand the benefits and detriments of each respective design concept, a decision matrix is a simplified way to aid the decision process. The results of the decision matrix are very useful in putting each concept on a level playing field in coordination with several key characteristics. Each design concept was rated on a scale of 1-4, with 4 being the best, with respect to its potential cost, efficiency, reliability, and practicality. These characteristics are then weighted by importance, with cost and efficiency being weighted more so than reliability or practicality. Each concept’s characteristic rank is multiplied by the weight factor, and then summed to a total where the highest score is the “best” design.

Multiple possibilities for recouping lost power were researching over the course of the project. In order to verify that the Closed Loop Power circuit was the most desirable method, a decision matrix was created to compare the four options. It was decided that cost, efficiency, reliability, and practicality were the four most important factors in determining the quality of the system. Although start-up cost will eventually be recovered by the cost savings generated by the system, it must still be taken into account as an initial investment by the company. The accumulator
The circuit is the lowest cost since little additional equipment is needed. The generator circuit is the most expensive due to the high cost of the grid-tie inverter and electric generator.

Efficiency is a measurement of how much of the lost energy is being recovered. A system based on recovering energy must take efficiency into account, as a more efficient system will recover more power and therefore conserve more energy. The accumulator circuit is the most efficient since the energy remains in the fluid, and accumulators are highly efficient devices. The heat recovery system, as outlined in the heat recovery section, is the least efficient since it only recovers a fraction of the lost power and is only useful during the cold months.

Reliability is a concern for laboratory equipment as less reliable systems require more attention from technicians and also pose a greater safety risk. The closed loop power circuit is the most reliable (and safe) since all of the components are contained within the pump/motor enclosure. The hydraulic motor will convert to a pump when no flow is supplied, preventing the motor from cavitating. The least reliable system is the accumulator series circuit, since accumulators require discharging before and after use. A charged accumulator can pose safety hazards to the technicians if not properly maintained. The nature of durability testing requires regular disassembly and reassembly of the test system, and since the accumulators are plumbed directly into the system, they must be removed at each test interval.

Practicality is the final factor in comparing the four recovery concepts. The closed loop power circuit is the most practical since it can be applied to any system and will recover power as long as the system is running. It can be applied to any test stand in which a load must be simulated. The accumulator series circuit is the least practical since it can only be used when multiple pieces are being cycled, and it greatly increases the cycle time required (since pressure and flow must be passed from one valve to the next through the accumulator, valves cannot be cycled into the open position simultaneously).

As seen in Table 3, the Closed Loop Recovery Circuit scored the highest point total among design alternatives. This can be largely attributed to its strength in all categories and lack of a definitive weakness, whereas the other alternatives had at least one glaring flaw. The Heat Recovery design was limited by its lack of efficiency, reliability, and practicality. The Accumulator Series Pressure Circuit, though efficient and cost effective, could not make up enough ground due to the lowest scores in reliability and practicality. The Hydraulic Generator Circuit scored low in the higher weighted categories of cost and efficiency, ultimately limiting its point total. The results from the decision matrix confirm the conceptual reasoning that the Closed Loop Recovery circuit is the most attractive design concept on which to conduct further analysis.
<table>
<thead>
<tr>
<th>Decision Matrix</th>
<th>Cost</th>
<th>Efficiency</th>
<th>Reliability</th>
<th>Practicality</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight Factor</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.2</td>
<td>1</td>
</tr>
<tr>
<td>Heat Recovery</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Accumulator</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>2.8</td>
</tr>
<tr>
<td>Generator Circuit</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2.1</td>
</tr>
<tr>
<td>Closed Loop Power</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3.1</td>
</tr>
</tbody>
</table>

Table 3: Decision Matrix for hydraulic test stand design alternatives.

**AMESim Model**

AMESim, a 1-D system modeling software, was used to simulate the final design and validate and quantify the effectiveness of the closed loop recovery system. AMESim was chosen because it is able to incorporate multiple physical domains. In this case, electrical, rotary, and hydraulic components were used to recreate the system.

Slight changes were made from the circuit seen in Figure 15 to the modeled circuit shown in Figure 16 below. The differences between the design concept and model are outlined below in the Model Component section.

---

**AMESim Simulation Components**

**Motor Signal Input**

The motor signal is used to set the speed of the electric motor driving the pump. This is a piecewise signal; however the motor speed is constant in most applications, so only one setting is needed. The input parameter into the signal is 1800 (unitless).
Variable Speed Prime Mover

The prime mover is used to simulate the electric motor that provides power to the pump. It receives a null signal from the signal input and converts it into power. The speed is controlled by the signal input – in this simulation it is set at 1800 RPM.

Rotary Power Sensor

The power sensor is used to convert the rotary motion of the electric motor into power applied to the pump. This subcomponent is used to convert rotary motion into power, and no input is required.
Hydraulic Pump

The hydraulic pump is the central component of any hydraulic test cell. This simulation uses a fixed displacement hydraulic pump; in this case the pump displacement is 50cc (3.05in³). At a constant speed of 1800 RPM, the ideal flow rate is the product of displacement and speed, or 90 LPM (23.78 GPM). In addition to pump displacement, the mechanical and volumetric efficiency of the pump is defined. For the purpose of this simulation, the mechanical and volumetric efficiency of the pump is set at a constant 90% (0.9). The mechanical efficiency loss means more power must be consumed to keep the motor spinning at 1800 RPM. The 90% efficiency estimate is taken from the efficiency plot in Figure 13, assuming a constant speed of 1800 RPM and 150bar system pressure. The volumetric efficiency loss assumes a 10% loss of fluid volume through pump leakage (case drain). This 10% loss means only 81 LPM (21.4 GPM) is circulated through the system.
The relief valve is the component being replaced by the closed loop recovery system. However, the relief valve is still required as a safety outlet and to control system pressure. Ideally, there would be no flow across the relief valve, indicating the hydraulic motor is providing exactly the load required. If the motor is undersized, some flow will dump across the relief in order to maintain the required pressure in the line. Since the load required by the test setup is 150bar, the relief valve will be set to maintain 150bar. Node 2 is in line with the recovery circuit, and node 1 routes back to the test cell's reservoir. Flow routed across the relief valve to maintain system pressure will be dumped back into the cell's reservoir.
Flow Control Valve (3-way)

The flow control valve pictured in figures 25 and 26 below represents the valve being tested in the system. Within the recovery loop, this can be interpreted as the “on/off switch” of the circuit. The flow from the pump enters at node 1 (P). When the valve is open, the flow goes through node 3 (A) and into the hydraulic motor. When closed, all flow is routed through node 2 (T) into the reservoir. The valve’s spool position is controlled by an electrical current input, which in this simulation is modeled as a signal input. The input signal in this case is stepped from 1 to 0 and back to 1 in 3-second intervals. This makes it possible to compare the system behavior when the circuit is active and inactive without creating two separate simulations. The spool position varies linearly from fully closed (routed to T) with an input of 0 to fully open (routed to A) with an input of 1. With “x” representing the null signal input, the maximum flow rate through the valve is set to 100 LPM (26.42 GPM) to account for the maximum flow supplied by the pump.
Tank (Reservoir)

At three separate locations in the simulation circuit, flow is routed to an open tank. In AMESim, a tank is modeled as a constant pressure source; in this case the pressure is 0bar since the tank is open. Flow through the relief valve is routed back to the tank; in the original circuit this would be 100% of the fluid. However, this relief valve is used to limit pressure in the line, so fluid is dumped across the relief valve into the tank as necessary to maintain 150bar in the relief circuit. The tank port of the 3-way valve is also routed to tank. In the case of this simulation, this flow path represents the recovery system being “inactive”, since all flow is routed to the tank and no flow enters the hydraulic motor. The third tank used in the simulation is downstream of the hydraulic motor. Data pulled from the hydraulic motor shows a 150bar pressure drop across the motor, verifying that kinetic energy from the fluid has been consuming in creating torque in the hydraulic motor.

Hydraulic Motor

A hydraulic motor is essentially a hydraulic gear pump in reverse. Instead of applying power to a fluid, as in a pump, a fluid applies power to the motor to generate power. Fluid carrying a certain amount of energy (product of pressure and flow) enters the casing of the hydraulic motor and turns
the gears within the hydraulic motor, which are connected to a shaft in which torque is generated. Sizing of the hydraulic motor will be discussed in the simulation results section. The motor used for the final results has a displacement of 35cc (2.14in³). Assuming a volumetric efficiency of 90%, as with the pump, the motor has a maximum flow capacity of 63 LPM (16.64 GPM).

The hydraulic circuit simulated using AMESim can be seen in Figure 16 and is very similar to the schematic of the actual circuit. The pump in the AMESim circuit had a displacement of 50 cc and was set at a constant speed of 1800 rpm. Table 4 gives a detailed assessment of the variance of motor displacement and the resultant pressure and flow measurements at the pump itself, the relief valve before the customer directional control valve, and the hydraulic motor designed to capture excess energy. This energy transaction is also detailed in Table 4, where the output of mechanical power is listed with the changing hydraulic motor displacement. This data is used to establish a relationship between a hydraulic pump and the displacement of a hydraulic motor that could potentially be used to recycle excess energy.

**Figure 28: Hydraulic Motor**

**Figure 29: Hydraulic Motor Parameters**

**Program Output**

The hydraulic circuit simulated using AMESim can be seen in Figure 16 and is very similar to the schematic of the actual circuit. The pump in the AMESim circuit had a displacement of 50 cc and was set at a constant speed of 1800 rpm. Table 4 gives a detailed assessment of the variance of motor displacement and the resultant pressure and flow measurements at the pump itself, the relief valve before the customer directional control valve, and the hydraulic motor designed to capture excess energy. This energy transaction is also detailed in Table 4, where the output of mechanical power is listed with the changing hydraulic motor displacement. This data is used to establish a relationship between a hydraulic pump and the displacement of a hydraulic motor that could potentially be used to recycle excess energy.
The goal of the AMESim simulation was to visualize the effects of a variable hydraulic motor in a hydraulic circuit designed to recoup excess energy, specifically the pressure, flow, and mechanical power. From these visualizations, conclusions about the most efficient hydraulic motor displacement can be drawn, thus allowing for theoretical numbers to compare with experimental results.

As mentioned before, these effects were recorded at the pump, the relief valve, and the hydraulic motor included in this circuit and the results can be seen in Figure 30 and Figure 31. Figure 30 displays how increasing motor displacement, in terms of percent of the pump displacement (i.e. a 5 cc hydraulic motor corresponds to 10% of the pump’s 50 cc displacement), changes the flow of the specifically mentioned components in the circuit. Pump flow is maintained at a constant 81 L/min. Relief and motor flow are inversely related, with motor flow increasing with motor size and relief flow decreasing with motor size. This result makes sense because as the motor increases, so does its ability to draw flow, thus less has to be relieved.
Figure 31 is similar to Figure 30 except that pressure, instead of flow, is measured against motor displacement. Relief pressure remains at zero for every change in the hydraulic motor since the relief is routed to an open tank. Pump and motor pressure stay fairly constant until the hydraulic motor is around 80% the size of the pump. It is at this stage that pump and motor pressure plummet to approximately 0-1 bar of pressure. This drop-off in system effectiveness is a result of the volumetric efficiencies of the pump and motor. In the simulation, a volumetric efficiency of 90% was assumed for the hydraulic pump. Since the line losses are dependent on the test piece and test setup itself, line losses are not factored into the simulation. The volumetric efficiency of the pump is also approximated at 90%. This means that 90% of the flow into the pump is forced through the pump outlet (the other 10% is lost to leakage). The same 90% approximation is used for the motor, since flow will also be lost to leakage in the motor casing. Accounting for both of these inefficiencies, 81% of the volume entering the pump is going through the motor.
In order to create power, the flow through the motor must be less than the flow through the pump. The differential in flow will create pressure at the motor inlet, which acts in place of the relief valve. If the flow into the motor is less than the motor capacity, the motor will act as a pump, and therefore no power will be created. Since the pump and motor are driven by a common shaft, the flow ratio will be equal to the displacement ratio (Flow = Displacement x Rotational Velocity), assuming equal volumetric efficiencies. At a 0.8:1 motor-pump ratio, it creates the maximum flow through the motor while still building pressure ahead of the motor. Since the motor is acting nearly perfectly as the relief valve at this point (approximately no flow through relief), the 0.8:1 ratio creates the maximum power recovery.

When the motor-pump ratio is increased to 0.82:1, the motor is drawing more flow than the pump output after volumetric inefficiency is accounted for. When this occurs, the motor will act instead as a pump since the flow input is less than the flow capacity of the motor. This is the central fundamental of sizing the hydraulic motor for the recovery system.
The simulation was also designed to compile data in regards to the mechanical input and output power. The input power would be the mechanical power used to power the pump of the circuit while the output power would be the mechanical power produced by the hydraulic motor at the end of the circuit. These parameters were also measured in accordance with variable hydraulic motor displacement and can be seen in Figure 32. From Figure 32, one can see that the 80% hydraulic motor size proves as the critical size for extreme behavior. Input power remains constant while output power steadily climbs until both drop to relative lows at the 80% mark (hydraulic motor stops generating torque).

The important conclusion that can be drawn from the figures created by the simulation is that there is a specific displacement for a hydraulic motor in this configuration that proves most efficient. Each of the figures indicates that, for the circuit design in this report, a hydraulic motor that is 80% of the circuit pump’s displacement will be able to generate the most mechanical output power. Motors that are below this threshold still will be able to generate mechanical power but at a proportionally lower rate. Motors that are above this threshold will not function correctly.

**Simulation Data**

It was determined that a motor equal to 70% of the pump displacement, would be used to gather data from the program. Although the maximum power is generated when the motor is 80% of the pump displacement, a margin of error is allowed to make room for system losses and minor fluctuations in flow.

The flow across the relief valve is an indicator of the system’s functionality. When the system is active (from 0<t<3 and 6<t<9) 10.3 LPM (2.72 GPM) is dumped across the relief. The remainder used by the hydraulic motor to generate power. When the system is inactive (3<t<6) the full 81 LPM (21.4 GPM) is dumped across the relief to build pressure in the line. Even with the motor at 70% of
the pump capacity, 87.3% of the flow supplied by the pump is used to generate power with the hydraulic motor.

Figure 33: Relief Valve Flow

The flow across the tank, pictured in Figure 34 below, shows the motor operating as a pump when the system is inactive. Since the motor is running constantly at 1800 RPM, with a displacement of 35cc, the flow capacity of the motor is 63 LPM (16.64 GPM). After accounting for 10% in efficiency losses, 56.68 LPM (14.97 GPM) is shown flowing from the tank into the hydraulic motor. The flow is negative in the graph since fluid is exiting the tank, not entering it. Figure 35 displays flow through the hydraulic motor. When the motor is loaded (system active), 70.7 LPM (18.68 GPM) is forced through the motor. When the system is inactive, the motor acts as a pump and flows the 56.68 LPM by drawing fluid from the tank.
The summing junction is used to measure the total power used by the electric motor during both active and inactive states. The power consumed during the active state is 10.946 kW (14.67 hp). The inactive state, which accounts for the electric motor driving the pump plus the electric motor, is 25.14 kW (33.7 hp). The additional power required to run the hydraulic motor is only needed to overcome the mechanical inefficiency of the motor, since there is no load on the motor during the inactive state. This result shows a power savings of 56.5% of the power consumed. In a system where the closed loop recovery circuit is applicable, it can cut the operating cost of the test in half.
The best way to understand the results of this project is to put them in terms of dollars and time saved. While it is impossible to predict how long the test stand would run or how much power it would draw, simple assumptions can be made to give a basic projection. As seen in the bill of materials in Table 5, the power recovery circuit is estimated to be approximately $3,910. This cost is based on the assumption that all components of the circuit will need to be purchased for installation with the 350 horsepower pump. In actuality, FEMA already has most of the components listed within its laboratory stock, but for the purposes of this cost analysis the projections were intended to cover the worst case scenario.

The next important factor in the cost analysis is the cost of the electricity needed to run the hydraulic test stand. First, the cost to run the test stand without power recovery components must be calculated. Assuming that the test stand runs eight hours a day for 365 days with an industrial rate of electricity in Michigan running at $0.065 per kWh, the cost of running the test stand comes to $49,537.90. As seen from the previous sections in this report, the potential power that can be recovered results in an approximate energy savings of 50%. This translates to a cost reduction of $24,768.90. If the circuit runs with the power recovery configuration and the same assumptions, the time to recover the initial investment for the necessary components amounts to approximately 59 days. This cost analysis confirms that the hydraulic power recovery circuit is a worthwhile investment both in terms of engineering and business sense.
Conclusion

Appropriate analysis of the options surrounding capital investment within a company is a fundamental approach to good business. With FEMA’s purchase of a 300 horsepower pump comes the opportunity to expand their customer base greatly, but only if the pump is used in such a way that its power potential is not wasted on a day to day basis. This report has detailed four specific configurations rooted in the theory of energy conversion with the goal of recovering enough energy to optimize the use of the 300 horsepower pump. The concept of heat recovery, though simple, proved to waste enough energy on its own to make the concept not feasible. The Accumulator Series Pressure Circuit, though cheap and efficient, proved to be vastly more complicated and specific than the scope that this project was willing to go. The Hydraulic Generator Circuit is attractive because of its versatility and electric cost reduction, but does so at a very high capital investment and with efficiency losses.

Utilizing the thought process behind the Hydraulic Generator Circuit, the final design for further analysis became a closed loop recovery circuit. Rather than input the electricity from the test stand directly into the FEMA grid as the Hydraulic Generator Circuit would, the power is used locally and inputted directly into the test loop. In this configuration, the motor acts as a pump when the system is not engaged, thus reducing the current being drawn from the electric motor. As shown from the decision matrix, the closed loop system proved to be the best approach to the problem in comparison to the previously mentioned solutions, owing largely to its fewer moving parts, higher efficiencies, and compact construction.

To verify the conceptual energy reduction proposed through thermodynamic logic, the Closed Loop Recovery Circuit was then modeled in AMESim, a fluid flow simulation program. The simulation tracked pressure and flow at various components throughout the circuit, as well as power input and output, as the hydraulic motor increased in size. Because the circuit’s effectiveness is limited mostly by the size of the hydraulic motor, the variable status of the size of the motor was designed to

<table>
<thead>
<tr>
<th>Stock/Part Number</th>
<th>Catalog</th>
<th>Part Name</th>
<th>Quantity</th>
<th>Cost</th>
<th>BOM Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>59595K12</td>
<td>McMaster-Carr</td>
<td>Bladder Style Accumulator</td>
<td>1</td>
<td>$723.15</td>
<td>1 gal capacity, 150 gpm max</td>
</tr>
<tr>
<td>5843T25</td>
<td>McMaster-Carr</td>
<td>Sure-Seal Steel Check Valve</td>
<td>2</td>
<td>$49.91</td>
<td>Pipe is 3/8 in., Male inlet/outlet, 3000 psi max</td>
</tr>
<tr>
<td>5PZK4</td>
<td>Grainger</td>
<td>Eaton 103-1573-012 Hydraulic Motor</td>
<td>1</td>
<td>$489.25</td>
<td>3.6 cu in/rev, 2500 psi max</td>
</tr>
<tr>
<td>4UA85</td>
<td>Grainger</td>
<td>Eaton Relief Valve, 7/8-14 In UNF-2B Port</td>
<td>1</td>
<td>$446.25</td>
<td>45 gpm max flow, 1500-3000 psi range</td>
</tr>
<tr>
<td>4CFX8</td>
<td>Grainger</td>
<td>Ashcroft Digital Gauge/Transmitter</td>
<td>2</td>
<td>$511.00</td>
<td>3 in dial size, 3000 psi</td>
</tr>
<tr>
<td>4352K641</td>
<td>McMaster-Carr</td>
<td>High Accuracy Digital Flowmeter</td>
<td>2</td>
<td>$316.59</td>
<td>3-50 gpm range, 300 psi max</td>
</tr>
<tr>
<td>6136K414</td>
<td>McMaster-Carr</td>
<td>NEMA 143T/145T Base-Mount AC Motor</td>
<td>1</td>
<td>$462.84</td>
<td>3 phase, 3 hp, 3600 rpm</td>
</tr>
<tr>
<td>290S41</td>
<td>McMaster-Carr</td>
<td>Pump-to-NEMA Motor Adapter</td>
<td>1</td>
<td>$33.42</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>$3,909.91</strong></td>
<td></td>
</tr>
</tbody>
</table>
determine the optimal size for maximum energy recovery. Graphs of pressure, flow, and power reveal that the optimal hydraulic motor size is around 80% of the size of the circuit pump. With a steady input at a hydraulic motor this size, the power required from the system pump is reduced by 50%, depending on system friction and line losses. Further development will be needed to apply this theory and simulation to cyclic valve testing, but the simulation results satisfied the problem within the scope of this project.

**Project Continuation**

Several design stages need to occur for further development and ultimate implementation of the Closed Loop Recovery Circuit. The next stage includes the experimental testing of a scaled down model of the hydraulic test stand that FEMA is developing. This is an important step on two fronts, the most obvious hinging on the actual execution of the design. Once the scale model is confirmed to function as a reliable hydraulic circuit, lab testing will commence. Much as the use of the simulation was designed to verify theoretical results, the testing of a scale model will be used to compile laboratory data in order to verify the results of the simulation. A confirmation from experimental data allows for full scale implementation and further testing, with the ultimate goal of continual use.

Another consideration regarding the Closed Loop Recovery Circuit is its application to cyclic testing. In this setting, the use of an accumulator would be an ideal component addition because of its capacity to balance pressure spikes. Cyclic testing also presents problems for the motor while it is inactive, during which it would act as a pump. This is a problem because not only does it consume energy, but fluid flow may be disturbed which may alter valve test results. A possible remedy for this dilemma is the addition of a sprag clutch, which would decouple the hydraulic motor from the electric motor when no torque is applied.

Though there is still plenty of work to be put into the project in the future, there is a definitive design procedure that is in place for execution. Both experimental results and cyclical testing additions are not outside standard hydraulic testing practice, and will lead to an effective and efficient high power hydraulic test stand at FEMA Corporation.
Bibliography

- Grainger. 100 GRAINGER PARKWAY, Lake Forest, Illinois, 60045. 
  <http://www.grainger.com/Grainger/wwg/start.shtml>
- Leugner, Tex. "Troubleshooting Hydraulic System Problems." Maintainence Technology 
  <http://fileserver.urimarketing.com/LU/CourseCatalog/Tex_Leugner_Hydraulic_Troubleshooting.pd f>.
- Parr, Andrew. Hydraulics and Pneumatics: A technician's and engineer's guide. Oxford: Newnes, 
  Print.
- Training Department, Motion & Control Group. Fluid Power Basics. Cleveland: Parker Hannifin, 
Appendix A: Personnel

Industrial Mentor
  o Jeff Huffman
    • Contact Information
      o FEMA Corporation
      o 1716 Vanderbilt Ave
      o Portage, MI 49028
      o jeffhuffman@fema-corp.com
      o 269-492-6002

Faculty Mentor
  o Christopher Cho
    • Contact Information
      o Western Michigan University
      o 1903 West Michigan Avenue
      o Kalamazoo, MI 49008-5343
      o christopher.cho@wmich.edu
      o (269) 276-3422
Personnel Responsibilities

Student Personnel

• Spencer Mick
  o Major Role:
    ▪ Completed AMESIM simulation
    ▪ Evaluated efficiency breakdown of hydraulic motor-generator circuit & final circuit

• Peter Kanda
  o Major Role:
    ▪ Defined circuit components and design concepts
    ▪ Generate bill of materials and pricing of components

• Brian Doll
  o Major Role:
    ▪ Evaluate pros/cons of design concepts and alternative solutions
    ▪ Formatted final report

Professional Staff

• Jeff Huffman
  o Role:
    ▪ Provide assistance with quantifying the problem with the current design
    ▪ Provide assistance with applying the design concepts.
    ▪ Ensure the correct analysis is being completed.

Faculty Mentor

• Christopher Cho
  o Role:
    ▪ Provide assistance in understanding of thermodynamic processes
    ▪ Give critique of final report and presentation
PETER M. KANDA
peter.m.kanda@gmail.com
821 W. Kalamazoo Ave., Kalamazoo, MI  49007, 248.978.4723

EDUCATION
Bachelor of Science cum laude, Western Michigan University, Kalamazoo, MI
Major:  Aeronautical Engineering    Minor:  Mathematics    GPA:  3.41
Expected Date of Graduation: April 2012
Member of Lee Honors College, 2008 to present; Member of Tau Beta Pi Engineering Honors Fraternity;
Completed Fundamentals of Engineering Exam (EIT certified)

RELEVANT EXPERIENCE
Engineering Test Lab Intern            March 2011-Current
Parker Hannifin Corporation, Aerospace Group, Kalamazoo, MI
Assisted in the construction of test stands designed to test hydraulic pumps, motors, and system components under extreme conditions. Contributed to the creation of component test procedures. Exposure to various electric motors, hydraulic pump mechanics, and test console software (Labview) used for military and commercial aircraft applications.

Fuel Emission & Powertrain Controls Summer Intern            May 2010-August 2010
Eaton Corporation, Southfield, MI
Designed test fixtures and components using CAD program Solidworks. Used GD&T for specifying detail drawings. Assembled prototypes of solenoid valves in prototype lab using control plans and related processes. Exposure to DFMEAs and DV test plans used in a production released process. Visited several design facilities to gain exposure to multiple automotive and truck products. Visited a high volume manufacturing plant where efficient automated processes were deployed in the design and release of a product in a production environment.

Dishwasher/Cook            August 2010-March 2011
Davis Cafeteria, Western Michigan University, Kalamazoo, MI
Facilitated dishroom and other cafeteria duties while maintaining full-time student status.

Groundskeeper Technician            May 2009-August 2009
Somerset Collection, Troy, Michigan
Landscaping and general maintenance of grounds and equipment.

HONORS AND AWARDS
•  Dean’s List, 2008-2009
•  CEAS Scholarship, 2010
•  Dean’s Scholarship, 2008
•  Medallion Scholarship Finalist, 2008
•  Edward W. Stimpson “Aviation Excellence” Scholarship, 2008

SKILLS
Computer: Proficient in Microsoft Office programs, CAD (Solidworks), Minitab, LabVIEW software, Matlab, FEA experience.
Test procedure execution, soldering, test stand construction, test circuit construction exposure. Import/Export training, pump mechanics training, AC/DC motor training, GD&T exposure, DFMEA exposure, experience writing test procedures.
ACTIVITIES

Brian Allen Doll
1152 Elkhorn Lake Rd, Lake Orion, MI
248-342-4471
brian.doll33@gmail.com

Objective
To obtain a job related to aeronautical engineering that will further my experience in the field.

Education
2007- Present Western Michigan University, Kalamazoo, MI
Bachelor of Science in Aeronautical Engineering
Minor in Mathematics
Expected to graduate in April 2012
2003- 2007 Lake Orion High School, Lake Orion, MI
• Lamp of Learning three years
• 3.72 GPA

Extracurricular Activities
2004-2007 FIRST Robotics
• Designed and built functional components of a robot.
• Trained and mentored younger members.
• Publicized team achievements and met with local companies for fund raising.
• Team leader for division of fabrication team and oversaw build divisions.
• Operated lathe, drill press, and CNC machine.
• Inspected components for flaws

Work Experiences
Summer 2008 Goertz & Schiele Corp., Auburn Hills, MI
• Worked on General Motors V-8 engine line.
• Maintained and cleaned set of machines.
• Oversaw the operation of CNC machines in a production environment.

Summer 2010 Visioneering Inc., Fraser, MI
• Prepared Tooling Inspection Reports
• Created tooling ball/inspection tags are machined parts
• Oversaw final inspection on machined parts
Computer Skills

- Auto CAD
- Mechanical Desktop
- Microsoft Word, Excel and PowerPoint
- Inventor 10
- Matlab

Spencer J. Mick
Phone: (248)635-9146                821 W. Kalamazoo Ave
E-mail: spencer.j.mick@wmich.edu           Kalamazoo, MI 49007

Education: Western Michigan University, Kalamazoo, MI
Bachelor of Science in Engineering (Mechanical)
Expected graduation: April 2012
GPA 3.34
Minors: Math, Sociology

Employment: FEMA Corp.                January 2010 - Present
Field Service Engineering
- Evaluate and test customer and field returns; failure mode and root cause analysis
- Analysis of field use on electro-hydraulic products and components
- Write and submit customer reports in response to warranty claims

Component Manufacturing
- Design hydraulic test stand for automated honing cell
- Conduct tool life reliability and analysis studies to minimize cost
- CNC building floor plan and work cell design

Engineering Lab
- Prepare and conduct hydraulic testing for product engineers
- Write formal test reports
- Communicate test observations to requesting engineers

Cashier/Stock
- Operate and balance cash register
- Stock grocery and general merchandise

Computer Skills:
Microsoft Excel, Win-Smith Weibull, Minitab, Visual Basic, Mathcad, AutoCAD, LabVIEW, ANSYS, AMESim, SolidWorks

Experiences:
Lee Honors College: Member 2008-2012

Western Michigan Club Baseball: 2008-2012
- Treasurer 2010-2011: Manage team expenses and fundraising
- President 2011-2012: Organize all team events, manage communication with university and governing body

- Representative 2009-2011: Represent baseball team in university affairs
- Treasurer 2011-2012: Direct allocations for all sports clubs at Western Michigan University
- Registered student financial manager; Allocating body manager
Appendix B: ABET questions

Assessment of Program Outcome #9

ME 4790/ME 4800

The MAE faculty members have identified “A knowledge of contemporary issues” as one of the program outcomes for both mechanical and aeronautical engineering programs. Contemporary issues are any issues that you hear on the news related to new and old products and their safety, new innovations, technologies, standards and regulations in general. As you develop your proposal for your senior design project, we ask you to start answering the following questions. These questions would guide you in the development of ideas you need to include in your proposal and final project reports. You are required to submit the completed form with your final proposal in ME 4790 and again with your final report in ME 4800. In your proposal and report, please include page references in response to each question below.

Evaluation of program outcome “A knowledge of contemporary issues”

1. Why is this project needed now?

   This project is needed now because of the growing global awareness of clean energy and green technology. Not only is the concept behind the project better for the environment, but it also makes business sense. As we all know from our required Thermodynamics course, there is no such thing as a 100% efficient device, and the pump/test cell setup at the center of the project is no different. Our task is to harness the losses that occur from normal testing in the hope that less energy can be consumed by the company, affecting the company’s carbon footprint and energy bill in a positive light.

2. Describe any new technologies and recent innovations utilized to complete this project.

   Due to the specificity of the project, it is somewhat difficult to pinpoint new technologies or recent innovations that are utilized in this project. The main model that our project will most likely follow will be similar to that of a four square test stand. This test stand can be summarized as a device that is used to test driveline components for a given
torque/power requirement without having to directly supply the test torque/power via a motor or engine.

3. If this project is done for a company—how will it expand their potential markets?
   ---how will it improve satisfaction of the company’s existing customers?
   ---identify the competitors for this kind of a product, compare the proposed design with the company’s competitors’ products.

   Our project does not deal directly with the company’s customers (Outcome #9, Question 3), so the project’s influence will be felt more by the company’s bottom line. There is the potential for money to be saved by lessening energy bills to run a test stand or multiple test stands. Obviously this money can be spent however the company sees fit, whether it is expanding the testing capabilities of the facility or hiring more talent. It is clear that since the project deals with the company’s existing equipment and is not part of their product line that there are no competing companies with whom we could compare designs to.

4. How did you address any safety and/or legal issues pertaining to this project (e.g., OSHA, EPA, Human Factors, etc.).

   Fortunately, no safety issues exist that are any different than what one would encounter within the company’s testing facilities. That being said, individuals unfamiliar with running a hydraulic test cell should not do so unless under the supervision of an experienced technician or engineer. Proper safety equipment, safety glasses, steel-toed shoes, etc. should be worn as well to prevent injury in the event something fails on the test stand.

5. Are there any new standards or regulations on the horizon that could impact the development of this project?

   There are not any new standards or regulations on the horizon that could impact the development of this project. The only relevance our project has with this question is the potential for the government to enforce some sort of energy consumption law in an effort to lower the amount of energy used to run the facility. Obviously this is an extreme case and in the event this would happen, the project would only help the company stay within some hypothetical energy limit.

There is the potential that the outcome of the project could result in a patent. As mentioned before, the four square test stand is the setup our project will attempt to follow and there is plenty of information related to this theory that can be useful.
The MAE faculty members have identified “**An understanding of the impact of the engineering solutions in a global, environmental and societal context**” as one of the program outcomes for both mechanical and aeronautical engineering programs. As you develop your proposal for your senior design project, we ask you to start answering the following questions. These questions would guide you in the development of ideas you need to include in your proposal and final project reports. You are required to submit the completed form with your final proposal in ME 4790 and again with your final report in ME 4800. In your proposal and report, please include page references in response to each question below.

**Evaluation of program outcome “An understanding of the impact of the engineering solutions in a global, environmental and societal context”**

1. Is this project useful outside of the United States? Describe why it is or is not—provide details.

   This project is useful outside of the United States because its end goal, power recovery, is a benefit for any engineering company. Power recovery is a universally praised endeavor that all countries strive to achieve. The design of the project may be specific to our company, but the thought process put into the design can be applied at any facility that needed to recover power wasted through from a pump mechanism to a test stand.

2. Does your project comply with US and/or international standards or regulations? Which standards are applicable?

   Our project does comply with US and/or international standard and regulations. Hydraulics testing is not specifically mentioned in OSHA standards definitions, but general test lab safety procedures are universally accepted. Our project also is in agreement with ANSI Standard T3.5.16-1991 as well as ASTM Standard T2.24.1-1991 and ASTM D6973 - 08e1, all of which deal with the hydraulics testing procedures.

3. Is this project restricted in its application to specific markets or communities? To which markets or communities?

   This project is restricted to the hydraulics market and markets that require fluid power for testing. Clearly any engineering firm that deals with fluid power can apply our project’s
design to a pump that is wasting energy and attempt to recover said energy, whether it is hydraulic oil or another kind of fluid.

4. If the answer to any of the following is positive, explain how and, where relevant, what were your actions to address the issues?

Design is focused on serving human needs. Design also can either negatively or positively influence quality of life. Address the impact of your project on the following areas:

**Air quality, water quality, food, noise level**

Does this project impact:

**Human health, wildlife, vegetation**

Does this project improve:

**Human interaction, well being, safety**

The nature of this project is different than that of some other projects in that we will not be creating a product for potential mass production. In fact, it will be confined to the company’s test facilities and will not directly affect the environment or surrounding citizens at all. Therefore, because the project is limited in scope, there will be no impact on air quality, water quality, food, noise level, human health, wildlife, and vegetation. This project also will not improve (or diminish) human interaction, well being, or safety.
Assessment of Program Outcome #13

ME 4790

The MAE faculty have identified “A recognition of the need for, and ability to engage in life-long learning” as one of the program outcomes for both mechanical and aeronautical engineering programs. As you develop your proposal for your senior design project, we ask you to start answering the following questions. These questions will guide you in the development of ideas you need to include in your proposal and final project reports, as well as help you identify areas in which you need improved proficiency. You are required to submit the completed form with your final proposal in ME 4790 and again with your final report in ME 4800 (addressing slightly different points of view). In your proposal and final project report, please include page references in response to each question below. This item will be included in the Team Assets section of the proposal. The format for the response to the questions in the report is of your own choosing but must address the below listed questions. Questions 2, 3 and 4 will also be directly addressed in the final Appendix of the report in the format shown below.

Your responses will be used in the Evaluation of program outcome “A recognition of the need for, and ability to engage in life-long learning”

A well organized team brings necessary backgrounds and talents together that are needed to successfully execute the design process. Each team member plays an important role on the design team. Individual members must be prepared to gain any additional skills necessary, and improve existing skills during project execution.

For each team member:
1. In detail identify the skills you bring to your design project that would be considered assets to the project team.

2. Delineate the skills necessary to successfully execute your responsibilities on the project.

3. Define skills you will personally need to strengthen to achieve the task at hand.

4. Explain how you plan to gain the skill level necessary to successfully execute your responsibilities to the design team.

Skills that Pete Kanda brings to the design project that would be considered assets to the project team are hydraulics testing experience acquired from internships at two hydraulics companies, and he is familiar with Matlab and some modeling software. The internship experience will be especially useful when testing flow control valves in order to obtain the data necessary for our project’s design. This data will then be analyzed, through the use of
spreadsheet software and other engineering software. Skills that Pete Kanda will need to strengthen will be knowledge of energy conversion, from fluid to mechanical, and the abilities to model computations in the correct engineering software. He plans to gain the necessary skill level to successfully execute his responsibilities by researching power recovery methods that are relevant to the project’s scope and use tutorials and manuals to learn computer programs that will be useful to the project.

Spencer Mick has two years of work experience with FEMA Corporation, and is therefore familiar with the company’s equipment and facilities in addition to general knowledge in hydraulic testing. He also has experience with instrumentation and data acquisition systems as well as data analysis. These skills will allow the team to acquire and analyze data from the prototype systems that will be developed throughout the course of the project. Familiarity with thermal design and energy conversion help to understand the background of the project, and mathematical modeling experience will allow the team to calculate theoretical results of the systems. He will need to improve skills in computational fluid dynamics in order to simulate potential solutions. This will be done by experimenting with ANSYS software.

Skills that Brian Doll brings to this group that would be considered assets are quality control experiences that have come from two years of internships at engineering companies, familiarity with different computer programs such as Matlab, EXCEL, Abaqus, AutoCAD, and Inventor. When completing tests of the designs, his experience in quality control with spotting and solving problems will help ensure everything runs smoothly. Brian Doll will have to become more familiar with identifying the thermal problems that occur when dealing with hydraulic processes of this scale. This skill will be gained through personal research and by learning from the industrial and faculty mentor.
Appendix C: Report Code

AMESIM Code
/******************** -*-C-*- *****************************/

* AMESim C code for ODE written by code generator. *

First_cut2_

*********************************************************************************/

#ifndef lint
static char vcid[] = "$Id: etemplate.c 11784 2010-07-30 07:28:48Z arne $";
#endif /* lint */

 ifndef lint
 static char vcid[] = "$Id: etemplate.c 11784 2010-07-30 07:28:48Z arne $";
 endif /* lint */

#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <signal.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#endif DEBUG
#include <assert.h>
#endif /*DEBUG*/

#include "ameutils.h"

static void EndOfSimulation(AMESIMSYSTEM *amesys);

#define MAX_FDRESULTS_BUFFER_SIZE (80 + NUMVARS)

static void fflush_fp_results();

extern int canHandlerLeave;

extern int shouldMainThreadLeave;

#ifdef WIN32

void setupieeefp(void); /* Make sure that the cpu is in the right state */

#include <io.h>

#define R_OK 4

#define pid_t int

#define SIGKILL 9

int kill(int pid, int signum)
{

return 1;
}

int getpgrp(void)
{
    return 1;
}

int getpid(void)
{
    return 1;
}

void setpgid(int pidno, int pidno2)
{
}

#endif

#ifdef SUNOS
#define killpg(p,s)  kill(-p,s)
#endif

ifndef SUNOS
#include <unistd.h>
#endif

static int first_call=1;
static char version[] = "LIB1.5.1";
static int statistics;

#define SUB_LENGTH       63
#define NUMSTATES        5
#define NUMSTATES2       25
#define NUMSTATESx13     65
#define NUMVARS           69
#define NUMDISCRETESTATES 0

/* Real parameters and stores. */
static double R0[4], R10[10]
    , R2[23], R19[4], R5[2], R6[2], R12[25], R12[9]
    , R14[25], R14[9], R15[6], R16[6], R17[4], R17[10]
    , R21[20], R22[7], R22[25], R25[5], R25[2]
    ;

/* Integer parameters and stores. */
static int   I0[5], I10[10]
    , I2[5], I10[10], I5[1], I6[3], I12[2], I12[2]
    , I14[2], I14[2], I15[1], I16[1], I17[5], I17[10]
    , I21[5], I22[2], I22[6], I25[2], I25[2];

/* Text parameters. */
static char   *TP0[2], *TP4[6], *TP17[2]
    , *TP21[2];

static double states[NUMSTATES];
static double v[NUMVARS];
static double vcopy[NUMVARS];
static double vprint[NUMVARS];
#if NUMDISCRETESTATES > 0
static double Z[NUMDISCRETESTATES];
#else
static double *Z = NULL;
#endif
static int points=0, iwrite, outoff;
static FILE *fp_results;
static int solvertype, stabiloption, minimaldiscont;
static int activityindex;
static int position;
static int HeaderSize;
static int NumToSave;
static int SaveFlags[NUMVARS];
static double *bufVar;
static int bufVarUsedSize;
static int bufVarAllocatedSize;
static int bufVarNextPos;
static double fixed_time;

static int consflag=0; /* Used for automatic assembly. */

/* These variables are used for the fixed time step solver */

static int FixedStep, fixed_type, fixed_order;
static double fixed_h;

/* *******************  Function prototypes ************ */

extern void pu001cin_(int *n, double *RP, int *IP, char **TP, double *RS, int *IS);
extern void pu001c_(int *n, double *ve0, double *ve1, double *ve2, double *ve3, double *ve4, double *vi5, double *vi6, double *vi7, double *vi8, double *RP, int *IP, char **TP, double *RS, int *IS);
extern void tk000in_(int *n, double *y0);
extern void hsv34in_(int *n, double *RP, int *IP, char **TP, double *RS, int *IS, double *y0, double *y1);
extern void hsv34_(int *n, double *ve0, double *ve1, double *ve2, double *ve3, double *ve4, double *ve5, double *ve6, double *ve7, double *ve8, double *ve9);
double *ve7, double *ve8, double *vi9, double *vidot9
, double *vi10, double *vidot10, double *vi11
, double *vi12, double *vi13, double *vi14
, double *vi15, double *vi16, double *vi17
, double *RP, int *IP, char **TP, double *RS
, int *IS, int *flag);
extern void sat0in_(int *n, double *RP, int *IS);
extern void sat0_(int *n, double *ve0, double *ve1, double *RP
, int *IS, int *flag);
extern void rv00in_(int *n, double *RP, int *IS);
extern void rv00_(int *n, double *ve0, double *ve1, double *ve2
, double *vi3, double *RP, int *IS, int *flag
);
extern void h3node3in_(int *n, double *y0, double *y1);
extern void ud00in_(int *n, double *RP, int *IP, double *RS, int *IS
);
extern void ud00_(int *n, double *ve0, double *RP, int *IP, double *RS
, int *IS, int *flag, double *t);
extern void pmv00in_(int *n);
extern void pmv00_(int *n, double *ve0, double *ve1);
extern void ptr00in_(int *n, double *RP, int *IP, double *y0, double *y1
, double *y2);
extern void ptr00_(int *n, double *port_1_v2, double *port_2_v1
, double *port_3_v2, double *int_v1, double *int_v2
extern double ptr00_output_(int *n, double *m0, double *m1, double *m2
, double *m3, double *m4, double *RP, int *IP, int *macindex
);
extern void ptr10in_(int *n, double *RP, int *IP, double *y0, double *y1
, double *y2);
extern void ptr10_(int *n, double *port_1_v2, double *port_2_v1
, double *port_3_v2, double *int_v1, double *int_v2
, double *int_v3, double *int_v4, double *int_v5
, double *int_v7, double *int_v6, double *int_dv5
, double *int_v7, double *int_dv7, double *RP
, int *IP);
extern double ptr10_output_(int *n, double *m0, double *m1, double *m2
, double *m3, double *m4, double *RP, int *IP, int *macindex
);
extern void mo001cin_(int *n, double *RP, int *IP, char **TP, double *RS
, int *IS);
extern void mo001c_(int *n, double *ve0, double *ve1, double *ve2
, double *ve3, double *ve4, double *vi5, double *vi6
, double *vi7, double *vi8, double *RP, int *IP
, char **TP, double *RS, int *IS);
extern void jun3min_(int *n);
extern void pm000in_(int *n, double *y0);
extern void ssinkin_(int *n);
extern void fp04in_(int *n, double *RP, int *IP, char **TP);
extern void hl03in_(int *n, double *RP, int *IP, double *RS, int *IS
    , double *y0, double *y1);
extern void hl03_(int *n, double *ve0, double *vedot0, double *ve1
    , double *ve2, double *vedot2, double *ve3
    , double *vi4, double *vi5, double *vi6, double *vi7
    , double *vi8, double *RP, int *IP, double *RS
    , int *IS, int *flag);
extern void hl000in_(int *n, double *RP, int *IP, double *RS, int *IS
    , double *y0);
extern void hl000_(int *n, double *ve0, double *vedot0, double *ve1
    , double *ve2, double *vi3, double *RP, int *IP
    , double *RS, int *IS, int *flag);

extern char **getssflist_(int *num_items)
{
    *num_items = 0;
    fprintf(stderr, "\nCall to getssflist possible only on real time platform.\n")
    AmeExit(0);
return NULL;
}

/*******************************************************************
*                                                                  *
*  This function returns the status of the assembly process.       *
*                                                                  *
*******************************************************************

extern int ConstructionLevel_()
{
    return consflag;
}

/*******************************************************************
*                                                                  *
*  It is sometimes useful to know the number of states. This       *
*  provides this information for an explicit system.              *
*                                                                  *
*******************************************************************

extern int GetNumStates()
{  
    return NUMSTATES;
}

/**
* This function returns the index in the given array of a given variable pointer
* Inputs: array - the array where the variable is stored.  
* array_size - size of the given array.  
* var - the variable we are looking for.  
* Return value: the corresponding index in the array.  
*/

static int GetIndex(double *array, int array_size, double *var)
{
    int i;

    for (i=0; i<array_size; ++i)
    {
        if (var == &(array[i]))
extern int GetStateVarIndex(double *var)
{
    return i;

}

{return 0;
return GetIndex(states, NUMSTATES, var);
}

extern int GetVarIndex(double *var)
{
    return GetIndex(v, NUMVARS, var);
}

/***************************************************************************/
*                                                                       *
*  This function returns the index in the v array of the given variable. *
*  This index corresponds to the rank of the variable in the results file.*
*                                                                       *
*  Inputs: var - the variable we are looking for.                       *
*                                                                       *
*  Return value: the corresponding index in the v array.               *
*                                                                       *
/***************************************************************************/
* finite is a replacement for the finite function that used to exist on HPUX and all other machines. HP seems to have removed it in HPUX11 replacing it with a macro isfinite. This function should make it possible to use the same libAME on both HP10 and HP11.

*******************************/

#ifdef HPUX

#ifdef isfinite

extern int finite(double value)
{
    return isfinite(value);
}

#endif

#endif

/*******************************/

* IsFixed returns True if the variable specified by the varnum is fixed.
extern int IsFixed(int var_num)
{
    static int num_fixed = 6;
    static int FIXED[6] = {11, 37, 0, 15, 30, 48};
    int i;
    for (i=0; i<num_fixed; ++i)
    {
        if (FIXED[i] == var_num)
        {
            return 1;
        }
    }
    return 0;
}

/**********************************************************************************
* firstc_ return True if its the first call to FunctionEval  else *
* it returns False. 
**********************************************************************************/
extern int firstc_()
{
    return first_call;
}

/********************************************************************
* PrintStateTitle reads the                                      *
* First_cut2_.state                                                 *
* file attempting to find a particular entry. If this is found, it* 
* is printed.                                                      *
* Input parameter:                                                  *
* int num position of entry in the file                            *
********************************************************************/
extern void PrintStateTitle(FILE *fptr, int num)
{
    FILE *fp;
    char buf[FILENAME_MAX+1];
    int i;

    if(num < 1 || num > NUMSTATES)
    {
        fprintf(stderr, "Call to PrintStateTitle ignored.\n");
        fprintf(stderr, "State number %d should be in the range 1 to %d\n",
                num, NUMSTATES);
        return;
    }

    fp = fopen(GetStateFileName(), "r");

    if(fp == NULL)
    {
        fprintf(stderr, "Can't open file %s\n", GetStateFileName());
        return;
    }

    i = 0;

    while(fgets(buf, FILELINE_MAX, fp) != NULL)
{ 
    i++; 

    if(i == num) 
    { 
        fprintf(fptr, " %s", buf); 
        break; 
    } 
} 
fclose(fp);

/*****************************************************************************/
/*                                                                  */
/*  AmeExit is the function that submodel and utilities should call  */
/*  instead of exit                                                   */
/*                                                                  */
/*****************************************************************************/
extern void AmeExit(int status)
{
    AMESIMSYSTEM *amesys = NULL;

    /* We are already exiting, no need to allow signal handler to call ameexit */
    canHandlerLeave = 0;
if(statistics)
{
    WriteRunStats();
}

EndOfSimulation(amesys);

AmeCallAtEnd(status);
AmeSignalModelUnload();

if (fp_results != NULL)
{
    fflush_fp_results();
    fclose(fp_results);
    fp_results = NULL;
}

ProcessTime(2);

if(status)
{
    fprintf(stderr,"AMESim model %s did an abnormal exit!\n", GetCircuitName());
fprintf(stdout,"AMESim model %s did an abnormal exit\n", GetCircuitName());
}
else
{
    fprintf(stdout,"AMESim model %s terminated normally\n", GetCircuitName());
}

if (bufVar != NULL)
    free(bufVar);
bufVar = NULL;
bufVarUsedSize = 0;
bufVarNextPos = 0;
bufVarAllocatedSize = 0;

exit(status);
}

/* This the equivalent function for use by Fortran programs */

extern void famexit_(int *status)
{
    AmeExit(*status);
}

Subroutine GetOldValues attempts to read an old file
First_cut2_.results
and reset state variables. If runtype=1, it will also attempt to
reset points and position for a continuation run.

22/11/02 STC numvars is always compared with NumToSave (even when it is > 0). We also check that the variables to save are the same as in the previous run.

17/04/09 OST 8897: warn user when cont run starts at final time.

static void GetOldValues(double *t, double *y, int *successful)
{
    /* Local variables. */
    long offset;
    int oldpts, numvar;
    int i, not_changed;
    int *saved_vars;
}
char *resultsname;

saved_vars = NULL;
not_changed = 1;
resultsname = GetResultsFileName();
*successful = 0;

if(access(resultsname, R_OK) != 0)
{
    /* File does not exist or exists but cannot be read. */

    return;
}

fp_results = fopen(resultsname, "r+b");

if(fp_results == NULL)
{
    return;
}

if ((fread( &oldpts, 1, sizeof(int), fp_results) != sizeof(int)) ||
    (fread( &numvar, 1, sizeof(int), fp_results) != sizeof(int)) ||
    oldpts < 1 ||
(numvar > 0 && numvar != NumToSave) ||
(numvar < 0 && numvar != -NumToSave))
{
    /* If the no. of variable whose values are being saved in *
    * results has been changed, it is impossible to perform *
    * continuation run without resizing the file header and *
    * moving data positions. Therefore, for the time being, *
    * abort execution. */

    fprintf(stderr, "Number of variables to be saved has changed,");

    if (isconrun_())
    {
        fprintf(stderr, "cannot perform continuation run.\n");
    }
    else
    {
        fprintf(stderr, "cannot use old final values.\n");
    }

    /* Close file and exit. */
    fclose(fp_results);

    fp_results = NULL;
    AmeExit(1);
else if (numvar < 0)
{

    /* Case where:
        (i) Not all of the variables are saved.
        (ii) The number of variables to save has not changed.
        The problem here, is that these variables may not
        be the same as in the previous run. */

    saved_vars = (int *)calloc(NumToSave, sizeof(int));

    if (fread( saved_vars, sizeof(int), NumToSave, fp_results) != NumToSave)
    {
        /* Error while reading the results file. */
        fclose(fp_results);
        fp_results = NULL;
        return;
    }

    for(i = 0; i < NumToSave; i++)
    {
        if (SaveFlags[saved_vars[i]] == 0)
        {
            not_changed = 0;
        }
    }
break;

}
numvar = -numvar;

offset = (long)(oldpts*(numvar+1)*sizeof(double)+(numvar+2)*sizeof(int));
}
else
{
    offset = (long)((oldpts-1)*(NUMVARS+1)*sizeof(double)+2*sizeof(int));
}

fseek(fp_results, offset, SEEK_SET);

if (isconrun_())
{
    /* Continuation run, we need the time */

    if(fread( t, 1, sizeof(double), fp_results) != sizeof(double))
    {
        /* Close file and return with successful false. */

        fclose(fp_results);
        fp_results = NULL;
        return;
    }
if (*t >= getfinaltime_())
{
    amefprintf(stderr, "Warning: continuation run with empty time interval\nstart time = %g s\nfinal time = %g s\n", *t, getfinaltime_());
}

} else
{
    double temp;

    /* Not a continuation run,
     we do not need the time variable read it into a dummy variable */

    if(fread( &temp, 1, sizeof(double), fp_results) != sizeof(double))
    {
        /* Close file and return with successful false. */
        fclose(fp_results);
        fp_results = NULL;
        return;
    }
}
if (fread((char *)v, sizeof(double), NUMVARS, fp_results) != NUMVARS)
{
    /* Close file and return with successful false. */
    fclose(fp_results);
    fp_results = NULL;
    return;
}

ChangeState(&y[2], v[20]);
ChangeState(&y[3], v[21]);
ChangeState(&y[0], v[2]);
ChangeState(&y[4], v[38]);
ChangeState(&y[1], v[17]);

if(isconrun_())
{
    /* Its a continuation run so must set points and position. */

    points = oldpts;
    position = points*(numvar+1);
    if(outoff)
    {
        fclose(fp_results);
        fp_results = NULL;
    }
else
{
    fclose(fp_results);
    fp_results = NULL;
}

/* Return with successful true. */

*successful = 1;
}

/**************************************************************************
*                                                                         *
*     InitSaveFlags - this function initialises the value of the           *
*                     output flags for each of the variables.              *
*                                                                         *
**************************************************************************/
if((fp = fopen(GetSaveFlagsFileName(), "r")) != NULL)
{
    i = n = 0;

    while(fgets(buf, FILELINE_MAX, fp) != NULL)
    {
        if((sscanf(buf, "%d", &SaveFlags[i]) == 1) && (SaveFlags[i] != 0))
        {
            n++;
        }

        i++;
    }

    fclose(fp);
}

else
{
    /* By defaults, the values of all variables are to be */
    /* written to the results file. */

    for(i = 0; i < NUMVARS; i++)
    {
}
SaveFlags[i] = 1;

)n = NUMVARS;
}

if(n < NUMVARS)
{
    HeaderSize = (2 + n) * sizeof(int);
}
else
{
    HeaderSize = 2 * sizeof(int);
}

NumToSave = n;

return n;
}

/***********************************************************
This function makes a copy of the v array.
extern void CopyVars()
{
    memcpy((void *)vcopy, (void *)v, NUMVARS*sizeof(double));
}

extern void RestoreCopyOfVars()
{
    memcpy((void *)v, (void *)vcopy, NUMVARS*sizeof(double));
}

This function restores the copy of the v array.

This function determines if any element of the v array has changed by at least 5 roundoff errors.
extern int AreVarsChanged()
{
    int i;
    double temp;

    for(i=0;i<NUMVARS;i++)
    {
        if(v[i] != vcopy[i])
        {
            temp = v[i] - vcopy[i];

            if(vcopy[i] + temp/5 != vcopy[i])
            {
                /* This variable has changed by at least 5 roundoff errors. */

                return 1;
            }
        }
    }
    /* No change. */

    return 0;
AmeReadFile opens the binary results file

First_cut2_.results

* If runtype\&1=0, it is a new run and the first records are
  * written to indicate the number of variables and that there
  * are zero points produced so far.

* If runtype\&1=1, a request has been made for a continuation run*
  * An attempt is made to read the old file the state variables,
  * points and position.

* If runtype\&2=2, a request has been made to use the final values*
  * of an old simulation as the starting values for a new. An
  * attempt is made to read the old file

First_cut2_.results

* and reset the state variables.

* If runtype & 4, it is some sort of stabilizing run and the file *
* is initialized as with runtype=0.                             *
*                                                                 *
*****************************************************************

static void AmeReadFile(double *t, double *y)
{
    /* Local variables */

    int successful;
    int i1, i2, i, nv;
    char line[PATH_MAX + 128];
    char *resultsname;

    resultsname = GetResultsFileName();

    /* Initialise the output value flags */

    nv = InitSaveFlags();

    if(isconrun_())
    {
        /* Its a continuation run */

        GetOldValues(t, y, &successful);
    }
if(!successful)
{
    /* Cannot do a continuation run. */

    ClearConRun();
}
else
{
    return;
}

else if (isusefinval_())
{
    /* New run with old final values as starting values */

    GetOldValues(t, y, &successful);

    if(!successful)
    {
        /* Cannot do run with old final values. */

        ClearUseFinalVal();
    }
}
if(outoff)
{
    return;
}
unlink(resultsname);

/* file has just been removed, create it and get Read/Write access */
fp_results = fopen(resultsname, "w+b");

if(fp_results == NULL)
{
    sprintf(line,"Cannot open results file `%s', resultsname);
    DisplayMessage(line);
}
i1 = 0;

if(fwrite(&i1, 1, sizeof(int), fp_results) != sizeof(int))
{
    sprintf(line,"Cannot write to file `%s', resultsname);
    DisplayMessage(line);
}
/* The value of the no. of variables must be negative */

(nv < NUMVARS) ? (i2 = -nv) : (i2 = NUMVARS);

if(fwrite(&i2, sizeof(int), fp_results) != sizeof(int))
{
    sprintf(line, "Cannot write to file \"%s\", resultsname);
    DisplayMessage(line);
}

if(nv < NUMVARS)
{
    for(i = 0; i < NUMVARS; i++)
    {
        if(SaveFlags[i])
        {
            fwrite(&i, sizeof(int), fp_results) != sizeof(int)
            {
                sprintf(line, "Cannot write to file \"%s\", resultsname);
                DisplayMessage(line);
            }
        }
    }
}
static void Input(double *y, double *tol1, double *tol2, 
               double *t, double *tfin, double *tinc, int *rstrtp, 
               int *statistics, int *mode, double *hmax) 
{
    /* Local variables */

    int error_type, idummy[9];
    double tol, rdummy[8];
    FILE *fptr;
    char text1[PATH_MAX+128], text2[PATH_MAX+128];

    /* Read First_cut2_.data to pick up submodel parameters */

    fptr = fopen(GetDataFileName(), "r");

    if(fptr == NULL)
    {

    }
sprintf(text1, "Unable to open file %s\n", GetDataFileName());
DisplayMessage(text1);
}

sprintf(text2, "Unable to read file %s", GetDataFileName());

sprintf(text2, "Unable to read file %s.\nCheck parameters of PU001C instance 1", GetDataFileName());
GetAndInterpretValues(fptr, RP0, 4, IP0, 5, text2);
GetAndInterpretTextValues(fptr, TP0, 2, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of TK000 instance 1", GetDataFileName());
GetAndInterpretValues(fptr, &v[11], 1, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of HSV34 instance 1", GetDataFileName());
GetAndInterpretValues(fptr, RP4, 23, IP4, 2, text2);
GetAndInterpretTextValues(fptr, TP4, 6, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of SAT0 instance 1", GetDataFileName());
GetAndInterpretValues(fptr, RP5, 2, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of RV00 instance 1", GetDataFileName());
GetAndInterpretValues(fptr, RP6, 2, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of TK000 instance 2", GetDataFileName());
GetAndInterpretValues(fptr, &v[37], 1, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of H3NODE3 instance 1", GetDataFileName());

GetAndInterpretValues(fptr, &v[0], 1, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of TK000 instance 3", GetDataFileName());

GetAndInterpretValues(fptr, &v[15], 1, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of TK000 instance 4", GetDataFileName());

GetAndInterpretValues(fptr, &v[30], 1, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of UD00 instance 1", GetDataFileName());

GetAndInterpretValues(fptr, RP12, 25, IP12, 2, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of PMV00 instance 1", GetDataFileName());

GetAndInterpretValues(fptr, RP14, 25, IP14, 2, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of UD00 instance 2", GetDataFileName());

GetAndInterpretValues(fptr, RP15, 6, IP15, 1, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of PTR10 instance 1", GetDataFileName());

GetAndInterpretValues(fptr, RP16, 6, IP16, 1, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of MO001C instance 1", GetDataFileName());
GetAndInterpretValues(fptr, RP17, 4, IP17, 5, text2);
GetAndInterpretTextValues(fptr, TP17, 2, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of JUN3M instance 1", GetDataFileName());

sprintf(text2, "Unable to read file %s.\nCheck parameters of PM000 instance 1", GetDataFileName());

GetAndInterpretValues(fptr, &v[48], 1, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of SSINK instance 1", GetDataFileName());

sprintf(text2, "Unable to read file %s.\nCheck parameters of FP04 instance 1", GetDataFileName());

GetAndInterpretValues(fptr, RP21, 20, IP21, 5, text2);
GetAndInterpretTextValues(fptr, TP21, 2, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of HL03 instance 1", GetDataFileName());

GetAndInterpretValues(fptr, RP22, 7, IP22, 2, text2);

GetAndInterpretValues(fptr, &y[0], 1, idummy, 0, text2);
GetAndInterpretValues(fptr, &y[4], 1, idummy, 0, text2);

sprintf(text2, "Unable to read file %s.\nCheck parameters of HL000 instance 1", GetDataFileName());

GetAndInterpretValues(fptr, RP25, 5, IP25, 2, text2);
GetAndInterpretValues(fptr, &y[1], 1, idummy, 0, text2);

fclose(fptr);

/* Read First_cut2_.sim to pick up simulation parameters */
fptr = fopen(GetSimFileName(), "r");

if(fptr == NULL)
{
    sprintf(text1, "Unable to open file %s\n", GetSimFileName());
    DisplayMessage(text1);
}

sprintf(text2, "Unable to read file %s", GetSimFileName());

GetValues(fptr, rdummy, 7, idummy, 9, text2);

*t    = rdummy[0];
*tfin = rdummy[1];
*tinc = rdummy[2];
*hmax = rdummy[3];
tol   = rdummy[4];

fixed_h   = rdummy[5];
fixed_order = (int)rdummy[6];

error_type = idummy[0];
iwrite     = idummy[1];
*rstrtp    = idummy[2];
*statistics = idummy[3];

ValidateRuntype(idummy[4]);

fixed_type = (int)(idummy[4] & 64) == 0;

solvertype = idummy[5];
stabiloption = idummy[6];
minimaldiscont = idummy[7];
activityindex = idummy[8];
outoff = 0;
*mode = 0;

fclose(fptr);

recordtinc_(tinc);

stabiloption += 4*solvertype;

RegisterRunParams(error_type, tol, *rstrtp, *statistics, activityindex,
  solvertype, idummy[4], iwrite, *tinc, *t, *tfin, *hmax);

if((int)(stabiloption & 4))
{
  

/* It is the cautious option. The maximum time step
    should not exceed the print interval. */

setmaxstep_(tinc);

if (error_type == 0)
{
    /* mixed (the default) error test */

    *tol1 = tol;
    *tol2 = tol;
}
else if (error_type == 1)
{
    /* relative error test */

    *tol1 = 1.0e-16;
    *tol2 = tol;
}
else
{
    /* absolute error test */

    *tol1 = tol;
*tol2 = 0.;

}

}

/***************************************************************************/
*
* This function calls the submodel pre-initialization *
* functions/subroutines to check resp, iesp and *
* state variable values and set the the con and icon *
* arrays *
*
***************************************************************************/
static void PreInitialize(AMESIMSYSTEM *amesys, double *y)
{
    int n=1;

}
static void Initialize(AMESIMSYSTEM *amesys, double *y)
{
    int n;

    n = 1;
    fp04in_(&n, RP21, IP21, TP21);

    n = 1;
    tk000in_(&n, &v[11]);

    n = 2;
    tk000in_(&n, &v[37]);

    n = 3;
    tk000in_(&n, &v[0]);

    n = 4;
    tk000in_(&n, &v[15]);

    n = 5;
tk000in_(&n, &v[30]);

n = 1;
ud00in_(&n, RP12, IP12, RS12, IS12);

n = 2;
ud00in_(&n, RP14, IP14, RS14, IS14);

n = 1;
pm000in_(&n, &v[48]);

n = 1;
sat0in_(&n, RP5, IS5);

n = 1;
pmv00in_(&n);

n = 1;
pu001cin_(&n, RP0, IP0, TP0, RS0, IS0);

n = 1;
rv00in_(&n, RP6, IS6);

n = 1;
ptr00in_(&n, RP15, IP15, NULL, NULL, NULL);

n = 1;
mo001cin_(&n, RP17, IP17, TP17, RS17, IS17);

n = 1;
jun3min_(&n);

n = 1;
ssinkin_(&n);

n = 1;
{

ptr10in_(&n, RP16, IP16, NULL, NULL, NULL);

}

n = 1;
hsv34in_(&n, RP4, IP4, TP4, RS4, IS4, &y[2], &y[3]);
n = 1;
h3node3in_(&n, &v[14], &v[34]);

n = 1;
hl03in_(&n, RP22, IP22, RS22, IS22, &y[0], &y[4]);

n = 1;
hl000in_(&n, RP25, IP25, RS25, IS25, &y[1]);
}

/***************************************************************
*                                                            *
* FunctionEval calls the submodels in an order that ensures that the  *
* inputs of each are known when it is called. When submodels flag    *
* discontinuities, conflicts are resolved.                     *
*                                                            *
***************************************************************/

define void FunctionEval(double *dot, double *y, double t, int *flag)
{
    int sflag, oflag, n, panic, i=0;
    static double zero = 0.0e0;
static int *oldflag, *newflag;

AMESIMSYSTEM *amesys = NULL; /* AJN temporary for testing */

/* Record old value of flag (oflag) and set
flag value for use in submodels (sflag).
Also get addresses of main discontinuity flags. */

oflag = *flag;

sflag = *flag;

if(first_call)
{
    GetFlagAddresses(&oldflag, &newflag);
}

/* Initialize everything ready for potential calls to stepdn
in submodels. */

panic = 0;

getredstep();

if(isstabrun_())
{
    t = fixed_time;
}

else if(*flag == 2)
{
    
    /* Record current simulation time for message passing. */

    SetSimTime(t);

}

    /* Record current simulation time for ametim_(). */

    SetTimeAtThisStep(t);

    if (holdinputs_())
    {
        
        /* We reset artificially the time to the initial value
        to give the illusion of held inputs. */

    
    t = getstarttime_();

    }

    /* Assign the state variables y[] calculated by the integrator
    to the appropriate variables v[]. */

    v[20] = y[2];
    v[21] = y[3];
    v[2] = y[0];
    v[38] = y[4];
\[ v[17] = y[1]; \]

/* Call submodel calculation subroutine in the order
   that ensures the input requirements of each submodel
   are satisfied. */

\[ v[40] = v[14] + v[34]; \]

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

n = 1;
*oldflag = *newflag = sflag;
ud00_(&n, &v[29], RP12, IP12, RS12, IS12, &sflag, &t);
if(sflag < 3)sflag = getnfg_();
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}
else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "UD00", 1, &panic);
}
n = 2;

*oldflag = *newflag = sflag;

ud00_(&n, &v[43], RP14, IP14, RS14, IS14, &sflag, &t);
if(sflag < 3)sflag = getnfg_();
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}
else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "UD00", 2, &panic);
}

v[52] = v[48] /* Duplicate variable. */;
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

n = 1;

*oldflag = *newflag = sflag;
sat0_(&n, &v[19], &v[29], RP5, IS5, &sflag);
if(sflag < 3)sflag = getnfg_();
if(*flag == 5)
{

LPerturbIfNecessary(flag);

else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "SAT0", 1, &panic);
}

n = 1;
*oldflag = *newflag = sflag;

pmv00_(&n, &v[42], &v[43]);

if(sflag < 3)sflag = getnfg_();
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}
else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "PMV00", 1, &panic);
}

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}
n = 1;

*oldflag = *newflag = sflag;

pu001c_(&n, &v[0], &v[3], &v[2], &v[5], &v[4], &v[6], &v[7], &v[8], &v[9], RPO, IPO, TPO, RS0, IS0);

if(sflag < 3)sflag = getnfg_();

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "PU001C", 1, &panic);
}


if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

v[12] = v[38] /* Duplicate variable. */;

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
v[32] = v[38] /* Duplicate variable. */;

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

n = 1;

*oldflag = *newflag = sflag;

{ /* DBK macro start. */
    int izero = 0;
    v[44] = ptr00_output_(&n, &v[5], &v[42], NULL, NULL, NULL, RP15, IP15, &izero);
} /* DBK macro end. */

if(sflag < 3)sflag = getnfg_();

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "PTR00", 1, &panic);
}

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

v[56] = v[17] /* Duplicate variable. */;

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

n = 1;

*oldflag = *newflag = sflag;

rv00_(&n, &v[31], &v[30], &v[32], &v[35], RP6, IS6, &sflag);

if(sflag < 3)sflag = getnfg_();

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "RV00", 1, &panic);
}

v[33] = -v[31] /* Duplicate variable. */;
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

n = 1;

*oldflag = *newflag = sflag;

{ /* DBK specific start. */

    ptr00_(&n, &v[5], &v[44], &v[42], &v[45], &v[46], &v[47], NULL, NULL, NULL, NULL, NULL, NULL, NULL, RP15, IP15);

}  

if(sflag < 3)sflag = getnfg_();

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "PTR00", 1, &panic);
}

n = 1;

*oldflag = *newflag = sflag;
mo001c_(&n, &v[36], &v[37], &v[51], &v[52], &v[56], &v[58], &v[59], &v[60], &v[61], RP17, IP17, TP17, RS17, IS17);

if(sflag < 3)sflag = getnfg_();
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}
else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "MO001C", 1, &panic);
}

v[57] = -v[36] /* Duplicate variable. */;
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

v[49] = v[51] /* Duplicate variable. */;
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

n = 1;
*oldflag = *newflag = sflag;
int izero = 0;

v[50] = ptr10_output_(&n, &v[48], &v[51], NULL, NULL, NULL,
    RP16, IP16, &izer0);
}

if(sflag < 3)sflag = getnfg_();
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}
else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "PTR10", 1, &panic);
}

v[62] = v[44] - v[50];
if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

n = 1;
*oldflag = *newflag = sflag;

{ /* DBK specific start. */
ptr10_(&n, &v[48], &v[50], &v[51], &v[53], &v[54], &v[55], NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, RP16, IP16);

if(sflag < 3)sflag = getnfg_();

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "PTR10", 1, &panic);
}

n = 1;
*oldflag = *newflag = sflag;

hsv34_(&n, &v[13], &v[12], &v[16], &v[15], &v[10], &v[11], &v[18], &v[17], &v[19], &v[20], &dot[2], &v[21], &dot[3], &v[22], &v[23], &v[24], &v[25], &v[26], &v[27], &v[28], RP4, IP4, TP4, RS4, IS4, &sflag);

if(sflag < 3)sflag = getnfg_();

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
else if(oflag != 5) {
    resdis(flag, &sflag, &oflag, "HSV34", 1, &panic);
}

if(*flag == 5) {
    LPerturbIfNecessary(flag);
}

n = 1;
*oldflag = *newflag = sflag;
hl03_(&n, &v[2], &dot[0], &v[3], &v[38], &dot[4], &v[39], &v[63],
    &v[64], &v[65], &v[66], &v[67], RP22, IP22, RS22, IS22,
    &sflag);
if(sflag < 3)sflag = getnfg_();
if(*flag == 5) {
    LPerturbIfNecessary(flag);
}
else if(oflag != 5) {
    resdis(flag, &sflag, &oflag, "HL03", 1, &panic);
n = 1;

*oldflag = *newflag = sflag;

hl000_(&n, &v[17], &dot[1], &v[18], &v[57], &v[68], RP25, IP25,
, RS25, IS25, &sflag);

if(sflag < 3)sflag = getnfg_();

if(*flag == 5)
{
    LPerturbIfNecessary(flag);
}

else if(oflag != 5)
{
    resdis(flag, &sflag, &oflag, "HL000", 1, &panic);
}

if(*flag == 0)
{
    /* It is an initialization call and the user
       is permitted to change the state variables
       and discrete variables. */

    ChangeState(&y[2], v[20]);
ChangeState(&y[3], v[21]);
ChangeState(&y[0], v[2]);
ChangeState(&y[4], v[38]);
ChangeState(&y[1], v[17]);
}
UpFECount();

if(first_call)
{
  /* Now is an excellent time to flush out any messages. */

  fflush(stderr);
  fflush(stdout);
  first_call =0;
}

/***************************************************************
*                                                                 *
* JFunctionEval is a 'cut down" version of FunctionEval for use when a *
* Jacobian is being evaluated.                                     *
*                                                                 *
****************************************************************/
extern void JFunctionEval(double *dot, double *y, double t, int col)
{
    int sflag, n=1, i=0;
    static double zero = 0.0e0;
    AMESIMSYSTEM *amesys = NULL; /* AJN temporary for testing */

    /* Only one flag value is required. */

    sflag = 1;

    /* Record current simulation time for ametim_. */
    SetTimeAtThisStep(t);

    if (holdinputs_())
    {
        /* We reset artificially the time to the initial value 
           to give the illusion of held inputs. */

        t = getstarttime_();
    }

    memcpy((void *)vcopy, (void *)v, (size_t)(NUMVARS*sizeof(double)));
/* Create the switch statement with a case for each state variable. */

switch (col)
{
    case 0:
        v[2] = y[0];
        n = 1;
        pu001c_(&n, &v[0], &v[2], &v[5], &v[4], &v[6], &v[3], &v[8], &v[9], &v[10], RP0, IP0, TP0, RS0, IS0);
        n = 1;
        { /* DBK macro start. */
            int izero = 0;
            v[44] = ptr00_output_(&n, &v[5], &v[42], NULL, NULL, NULL, RP15, IP15, &izer0);
        } /* DBK macro end. */
        n = 1;
        { /* DBK specific start. */
            ptr00_(&n, &v[5], &v[42], &v[45], &v[46], &v[47], NULL, NULL, NULL, NULL, NULL, NULL, NULL, RP15, IP15
        }

}
v[62] = v[44]-v[50];

n = 1;

hl03_(&n, &v[2], &dot[0], &v[3], &v[38], &dot[4], &v[39], &v[63], &v[64], &v[65], &v[66], &v[67], RP22, IP22, RS22, IS22, &sflag);

break;

case 1:
    v[17] = y[1];
    v[56] = v[17] /* Duplicate variable. */;

    n = 1;

    mo001c_(&n, &v[36], &v[37], &v[51], &v[52], &v[56], &v[58], &v[59], &v[60], &v[61], RP17, IP17, TP17, RS17, IS17);

    v[57] = -v[36] /* Duplicate variable. */;
    v[49] = v[51] /* Duplicate variable. */;

    n = 1;

    { /* DBK macro start. */
        int izero = 0;
        v[50] = ptr10_output_(&n, &v[48], &v[51], NULL, NULL, NULL, RP16, IP16, &izero);
    } /* DBK macro end. */

    v[62] = v[44]-v[50];
n = 1;
{
    /* DBK specific start. */

    ptr10_(&n, &v[48], &v[50], &v[51], &v[53], &v[54]
        , &v[55], NULL, NULL, NULL, NULL, NULL, NULL,
        , RP16, IP16);

    }

n = 1;

hsv34_(&n, &v[13], &v[12], &v[16], &v[15], &v[10], &v[11]
    , &v[18], &v[17], &v[19], &v[20], &dot[2], &v[21]
    , &dot[3], &v[22], &v[23], &v[24], &v[25], &v[26]
    , &v[27], &v[28], RP4, IP4, TP4, RS4, IS4, &sflag
    );


n = 1;

hl03_(&n, &v[2], &dot[0], &v[3], &v[38], &dot[4], &v[39]
    , &v[63], &v[64], &v[65], &v[66], &v[67], RP22, IP22
    , RS22, IS22, &sflag);

n = 1;

hl000_(&n, &v[17], &dot[1], &v[18], &v[57], &v[68], RP25
    , IP25, RS25, IS25, &sflag);

break;

case 2:
v[20] = y[2];

n = 1;

hsv34_(&n, &v[13], &v[12], &v[16], &v[15], &v[10], &v[11]
       , &v[18], &v[17], &v[19], &v[20], &dot[2], &v[21]
       , &dot[3], &v[22], &v[23], &v[24], &v[25], &v[26]
       , &v[27], &v[28], RP4, IP4, TP4, RS4, IS4, &sflag
    );


n = 1;

hl03_(&n, &v[2], &dot[0], &v[3], &v[38], &dot[4], &v[39]
      , &v[63], &v[64], &v[65], &v[66], &v[67], RP22, IP22
      , RS22, IS22, &sflag);

n = 1;

hl000_(&n, &v[17], &dot[1], &v[18], &v[57], &v[68], RP25
      , IP25, RS25, IS25, &sflag);

break;

case 3:

v[21] = y[3];

n = 1;

hsv34_(&n, &v[13], &v[12], &v[16], &v[15], &v[10], &v[11]
       , &v[18], &v[17], &v[19], &v[20], &dot[2], &v[21]
       , &dot[3], &v[22], &v[23], &v[24], &v[25], &v[26]
       , &v[27], &v[28], RP4, IP4, TP4, RS4, IS4, &sflag
    );

n = 1;

hl03_(&n, &v[2], &dot[0], &v[3], &v[38], &dot[4], &v[39]
, &v[63], &v[64], &v[65], &v[66], &v[67], RP22, IP22
, RS22, IS22, &sflag);

n = 1;

hl000_(&n, &v[17], &dot[1], &v[18], &v[57], &v[68], RP25
, IP25, RS25, IS25, &sflag);

break;

case 4:

v[38] = y[4];

v[12] = v[38] /* Duplicate variable. */;

v[32] = v[38] /* Duplicate variable. */;

n = 1;

rv00_(&n, &v[31], &v[30], &v[32], &v[35], RP6, IS6, &sflag
);

v[33] = -v[31] /* Duplicate variable. */;

n = 1;

hsv34_(&n, &v[13], &v[12], &v[16], &v[15], &v[10], &v[11]
, &v[18], &v[17], &v[19], &v[20], &dot[2], &v[21]
, &dot[3], &v[22], &v[23], &v[24], &v[25], &v[26]
, &v[27], &v[28], RP4, IP4, TP4, RS4, IS4, &sflag
);

n = 1;
hl03_(&n, &v[2], &dot[0], &v[3], &v[38], &dot[4], &v[39]
, &v[63], &v[64], &v[65], &v[66], &v[67], RP22, IP22
, RS22, IS22, &sflag);
n = 1;
hl000_(&n, &v[17], &dot[1], &v[18], &v[57], &v[68], RP25
, IP25, RS25, IS25, &sflag);
break;
}

memcpy((void *)v, (void *)vcopy, (size_t)(NUMVARS*sizeof(double)));
UpFECount();
}

/*****************************************************************
 * This function calls the submodel terminate functions/subroutines *
 * to allow for resource liberation etc *
 ******************************************************************/
static void EndOfSimulation(AMESIMS SYSTEM *amesys)
{ int n;

}  

/*******************************************************************/
/*                                                                 *
* OutputResults outputs numerical results if required             *
*                                                                  *
* (i) to the screen and/or                                        *
* (ii) to the file:- First_cut2_.results.                        *
*                                                                  *
*                                                                  *
*******************************************************************/
extern void OutputResults(double t)
{
    static double last_flush_stdout=0.0;
    static double last_flush_fdresults=0.0;

    /* Local variables */
    int has_flush_fd_results = 0;
    int i, old_used_size;
unsigned long int future_file_size;

double tt1;

/* NOTE:

bufVarUsedSize is supposed to be the number of values in bufVar. This integer is mainly used by

fflush_fp_results to write the buffer content.

bufVarNextPos is supposed to be represent the position in bufVar in which next values have to written.

This integer makes possible to overwrite continuation run values for instance, or values computed while

a stabilizing step. */

/* Function Body */

/* do not allow signal handler to call ameexit */

canHandlerLeave = 0;

if(outoff)
{
    /* No output required so nothing to do. */

    return;
}

/* get real time */

tt1 = AMEGetTime();

if(iwrite != 2 &amp;&amp; !staborcs_())
{
    /* Output time to screen. */
    fprintf(stdout, "Time = %g\n", t);
    if(tt1 - last_flush_stdout > 0.05)
    {
        fflush(stderr);
        fflush(stdout);
        last_flush_stdout = tt1;
    }
}

/* Write time & variables to First_cut2_.results. This is now
   done in one single write statement. We first fill the
   buffer array (buf) with the stuff we want to write */

/* allow values after bufVarNextPos to be overwritten */
bufVarUsedSize = bufVarNextPos;

/* check allocated size of bufVar */
if (bufVarUsedSize + 2 + NUMVARS + NumToSave > bufVarAllocatedSize)
bufVar = (double *)realloc(bufVar, (bufVarUsedSize + 2 + NUMVARS + NumToSave)*sizeof(double));

if (bufVar == NULL)
{
    amefprintf(stderr, "Not enough memory
Exiting...
");
    AmeExit(1);
}

bufVarAllocatedSize = bufVarUsedSize + 2 + NUMVARS + NumToSave;

old_used_size = bufVarUsedSize;

/* Fill the buffer with the normal writes */
bufVar[bufVarUsedSize] = t;
bufVarUsedSize++;
for(i=0;i<NUMVARS;i++)
{
    /* Write value to results file only if it is enabled. */

    if(SaveFlags[i] != 0)
    {
        bufVar[bufVarUsedSize]=v[i];
        ++bufVarUsedSize;
    }
/* If this is a selective save run (not all variables are saved) we
need (for a restart) to have a complete set of variable values
at the end of the file. This will be updated for each output
point so we must not update the position count here. */

if(NumToSave != NUMVARS)
{
    bufVar[bufVarUsedSize]=t;
    memcpy(bufVar + bufVarUsedSize + 1, v, NUMVARS*sizeof(double));
    bufVarUsedSize += 1 + NUMVARS;
}

future_file_size = (position+bufVarUsedSize)*sizeof(double)+HeaderSize;
if( future_file_size >= 0x80000000 )
{
    /* do not write values that just have been computed */
    bufVarUsedSize = old_used_size;
    amefprintf(stderr,"The simulation has been stopped before\n");
    amefprintf(stderr,"the results file exceeds the 2 GigaBytes limit.\n");
    amefprintf(stderr,"To avoid this problem, reduce the number of saved variables\n");
    amefprintf(stderr,"or increase the print interval.\n\n");
}
AmeExit(1);
}

/* Write number of points calculated. */
points++;

/* Write the buffer */
if (bufVarUsedSize > MAX_FDRESULTS_BUFFER_SIZE || tt1 - last_flush_fdresults > .5)
{
    fflush_fp_results();
    has_flush_fd_results = 1;
    last_flush_fdresults = tt1;
}

bufVarNextPos = bufVarUsedSize;
if(staborcs_())
{
    /* we do want that the values currently in bufVar to be overwritten next time */
    /* if these values have been written, update position so that they will be overwritten in resultsfile too */
    if (has_flush_fd_results)
    {
        position -= NumToSave + 1;
    }
}
/ * First position ourselves at the end of the data */

bufVarNextPos = old_used_size;

points--;

}
else if(has_flush_fd_results == 0 && NumToSave != NUMVARS)
{
/* allow continuation run values to be overwritten */

bufVarNextPos -= NUMVARS + 1;

}
#ifdef DEBUG
/* this function is supposed to exit with:

bufVarUsedSize == 0 if file has been flushed

bufVarUsedSize incremented with NUMVARS+1 if no need to save all values for continuation run

bufVarUsedSize incremented with NUMVARS+1 + NumToSave+1 if values for continuation run have been written */

assert((has_flush_fd_results && bufVarUsedSize == 0)
       || (NumToSave == NUMVARS && bufVarUsedSize == old_used_size + NUMVARS +1)
       || (NumToSave != NUMVARS && bufVarUsedSize == old_used_size + NumToSave + NUMVARS +2));

/* this function is supposed to exit with:

bufVarNextPos == 0 if file has been flushed

bufVarNextPos unchanged if we are in a stabilizing step

bufVarNextPos incremented with NumToSave+1 in the other cases */
assert((has_flush_fd_results && bufVarNextPos == 0) || (staborcs() && bufVarNextPos == old_used_size) || bufVarNextPos == old_used_size + NumToSave + 1);
#endif /*DEBUG*/

/* allow signal handler to call ameexit */

canHandlerLeave = 1;

/* if signal handler has been called */
if (shouldMainThreadLeave)
{
    /* exit */
    /* exit */
    AmeExit(2);
}
}

/*****************************************************************
*                                                                *
* fflush_fp_results writes values from bufVar into result file   *
*                                                                *
*                                                                *
*                                                                *
******************************************************************/

static void fflush_fp_results()
{  
  char line[PATH_MAX+128];

  /* nothing to do if bufVar is empty */  
  if (bufVarUsedSize == 0)  
    return;

  /* set file position at the end of the file so that next output can be done */  
  fseek(fp_results, (long)(position*sizeof(double)+HeaderSize), SEEK_SET);

  /* Write all values including the one for continuation run */  
  if(fwrite( bufVar, sizeof(double), bufVarUsedSize, fp_results) != bufVarUsedSize  
     || fflush(fp_results) != 0)  
    
    {  
      bufVarUsedSize = 0;
      sprintf(line,"Cannot write to file `\%s'", GetResultsFileName());
      DisplayMessage(line);
    }

  /* update header of the fp_results file with the right number of points */  
  fseek(fp_results, 0L, SEEK_SET);

  if(fwrite( (char *)&points, 1, sizeof(int), fp_results) != sizeof(int)  
     || fflush(fp_results) != 0)
bufVarUsedSize = 0;

sprintf(line,"Cannot write to file `%s'", GetResultsFileName());
DisplayMessage(line);

/* if continuation runs have been written, reset file position so that they will be overwritten next time*/
if(NumToSave != NUMVARS)
{
    /* bufVar contained continuation values, we must set position so that they will be overwritten */
    position += bufVarUsedSize - NUMVARS - 1;
}
else
{
    position += bufVarUsedSize;
}

/* buffer has been written: used size is now equal to zero */
bufVarUsedSize = 0;

/******************** *******************************************************************/
*/
* Output - deals with output of results. *

* Revisions: *

* 16/12/99 CWR The functions SetIsPrinting and ClearIsPrinting are called bracketing the call to res. This means that certain internal variables can be calculated only when a print is in progress. *

extern void Output(double t, double *y, double *dot, int *flag)
{
    static int oflag, cond;
    static double previous_t = -1.0e30;

    /* Function Body. */

    if(!outoff)
    {
        /* If no output is required, return */

        return;
    }

    oflag = *flag;
ProcessTime(1);

SetIsPrinting();

FunctionEval(dot, y, t, flag);

ClearIsPrinting();

/* If flag has changed, interpolation 'noise'
has generated an illegal state variable value
so ignore print */

if (*flag != oflag)
{
  return;
}

cond = !FixedStep && !staborcs_();

if (cond && veryclose(t, previous_t, 5))
{
  /* Very close to previous time. Determine if values are sufficiently
different to be worth adding to results file. */

  if (sametotol(v, vprint, SaveFlags, NUMVARS))
{ /* It is not worth adding this to the results file. */

    return;

} /* It is worth adding to the results file. */

OutputResults(t);

if(cond)
{
    previous_t = t;

    memcpy((void *)vprint, (void *)v, NUMVARS*sizeof(double));
}

}

static int ecount[NUMSTATES];

iscrimp((void *)v);

/**********************************************************************
*                                                                     *
*    upcount - maintains the ecount array. After each successful step *
*    the integrator calls upcount to increment the element of the     *
*                                                                     *
**********************************************************************/
extern void upcount(int num)
{
    ecount[num-1]++;
}

extern void WriteErrorCount(int close)
{
    int i;

    static FILE *errfp=NULL;
    static int file_closed=0;

    /* WriteErrorCount - writes the contents of the ecount array to a file. It is called by the integrator at regular intervals and at the end of the simulation run. In the latter case close=1 and the file is closed. */

    /******************************************************************************/
    addItemToCount();
    /************************************************************************--------*/

if(file_closed)
{
    return;
}

if(errfp == NULL)
{
    /* Open file for state errors after removing any old
       file of the same name. */

    unlink(GetErrorFileName());
    errfp = fopen(GetErrorFileName(), "w");
}

if(errfp != NULL)
{
    /* Rewind state errors file. */

    rewind(errfp);

    /* Write state errors file. */

    for(i=0;i<NUMSTATES;i++)
    {
        fprintf(errfp, "%d\n", ecount[i]);
    }
}
fflush(errfp);

if(close)
{
    fclose(errfp);
    file_closed = 1;
}

extern void SummarizeErrors()
{
    int i;
    FILE *fp;
    char line[FILELINE_MAX+1];
fp = fopen(GetStateFileName(), "r");

fprintf(stdout, "\n");

for(i=0;i<NUMSTATES;i++)
{
    fprintf(stdout, "State %2d controlled step %5d times\n", i+1, ecount[i]);

    if(fp != NULL)
    {
        if(fgets(line, FILELINE_MAX, fp) != NULL)
        {
            fprintf(stdout, "    %s", line);
        }
        else
        {
            fclose(fp);
            fp = NULL;
            fprintf(stdout, "\n");
        }
    }  

else
{
    fclose(fp);
    fp = NULL;
    fprintf(stdout, "\n");
    
}  

else

{  
    fprintf(stdout, "\n");  
}

if(fp != NULL)
{
    fclose(fp);
}

**********************************************************************
*                                                                     *
*     GetVar - this function services a request for a particular      *
*              variable number. The function is called during linear  *
*              analysis.                                              *
*                                                                     *
**********************************************************************

extern double GetVar(int num)
{
    return v[num];
}

/*******************************************************************************/
SetVar - this function sets a specified member of the variable array to a specified value. The function is called during linear analysis.

```
extern void SetVar(int num, double value)
{
    v[num] = value;
}
```

These functions terminate or kill the program group that the current process belongs to. THEY SHOULD NOT BE CALLED DIRECTLY. Arrange for them to be called by using the setAMEterminate function. This will make sure they are called when ever there is a call to exit.

The soft kill function AMEterminate first arranges that the current process ignores the signal sent to the rest of the group. After this, the process sends a kill process group (killpg). This should normally kill any children (or parents). The current process is about to die anyway.
By ignoring the signal we ensure that any other clean-up functions registered will be executed. We also allow for further customization in the future.

The hard kill function AMEterminate kills the group immediately.

Function author:

A. Jansson

Date written:

06/05/98

This functions calls:

signal
killpg or kill
getpgrp

This function is called by:

setAMEterminate

Function inputs:

-

Function outputs:

-

Revisions:
static void AMEterminate(void) 
{
    /* It is possible to insert statements below this line. */

    /* First protect ourself against the signal 
     * we will send to the group. */

    signal (SIGTERM, SIG_IGN);

    /* Now send a signal to the group to make them 
     * terminate. */

    killpg(getpgrp(), SIGTERM);

    /* It is possible to insert statements below this line. */

}

static void AMEsigkill(void)
{

/* It is possible to insert statements below this line. */

/* Now send a signal to kill the group. */

`killpg(getpgrp(), SIGKILL);`

`}`

/**************************************************************************/

The function `setAMEterminate` arranges for the program group to be
terminated or killed when this program exits.

The brutal parameter decides the way the group should be killed.

If brutal is false (0), the program group is terminated
which might give the participating programs a chance make a graceful
exit.

If howbrutal is true (1), the program group is KILLED.

Insert the following in a part of your code that is executed ONCE:

`setAMEterminate(1);`  for a sure and ugly kill
setAMEterminate(0); for a nice and friendly kill

This function is NOT callable from Fortran.

Function author:

A. Jansson

Date written:

06/05/98

This functions calls:

atexit

This function is called by:

main

Function inputs:

int brutal  1 for a hard kill and 0 for a soft kill

Function outputs:

-

Revisions:

******************************************************************/

static void setAMEterminate(int brutal)
if (brutal)
{
    /* Hard kill. */

    atexit(&AMEsigkill);
}
else
{
    /* Soft kill. */

    atexit(&AMEterminate);
}

******
* isactivity_ returns true if this option is selected in the Run parameters and isprint_ returns true. It returns false otherwise. *
******

extern int isactivity_();
{

int main(int argc, char **argv)
{
    /* Local variables, the big arrays are made static
to put them in the heap, avoiding stack overflow on PCs */

    static double acor[NUMSTATES], savf[NUMSTATES],
        atol[NUMSTATES], rtol[NUMSTATES],
        yh[NUMSTATESx13], ewt[NUMSTATES];
    char *batch_extension;
    double t, hmax, tinc, tol1, tol2;
    static int fixed[NUMSTATES];
    int i, rstrtp, mode, success;
pid_t pidno;

double start_time, final_time;

AMESIMSYSTEM *amesys = NULL; /* AJN TODO */

bufVar = NULL;
bufVarUsedSize = 0;
bufVarNextPos = 0;
bufVarAllocatedSize = 0;

/* Set group id to the same as the process id. */

pidno = getpid();

setpgid(pidno, pidno);

/* install handler for signals such as SIGINT */
installCallAMEExitOnSignalEvent();

/* Handle argument list (if any). */

batch_extension = NULL;

for (i=1; i<argc; ++i)
{

if (strcmp(argv[i], BATCH_OPTION_MARKER) == 0 && argc > i)
{
    batch_extension = argv[i+1];
}

/*@ 
* Use a limited buffer on stdout and no buffer for stderr.
* /

setvbuf(stdout, NULL, _IOFBF, 256);
setvbuf(stderr, NULL, _IONBF, 256);

/*@ Install an function to be called at exit. */

setAMEterminate(0);

/*@ Check that the correct library is installed. */

if(!CheckLibraryVersion(version))
{
    AmeExit(1);
}

/*@ Use argument list to create filenames. */
ConstructFileNames(argv[0], batch_extension);

/* Initialize error handles */
#ifdef WIN32
    ErrorHandlers();
#else
    setupieeefp();
#endif

/* Call Input to read submodel and simulation parameters. */
Input(states, &tol1, &tol2, &start_time, &final_time, &tinc, &rstrtp,
       &statistics, &mode, &hmax);

/* Call pre-initialize function */
PreInitialize(amesys, states);

if( NeedReloadInputFiles() != 0 )
{
    ClearGPs();
    Input(states, &tol1, &tol2, &start_time, &final_time, &tinc, &rstrtp,
&statistics, &mode, &hmax);

ClearReloadedFiles();
}

t = start_time;

if(FixedStep)
{
    int fixedsolvertype;

    /* The solvertype is coded with 100 for AB, 200 for RK added to the order */
    fixedsolvertype = ( fixed_type == 1)*100 + (fixed_type != 1)*200 + fixed_order;

    SetIsUsingFixedSolver(fixedsolvertype);
    SetFixedTimeStep(fixed_h);
}

/* Call initialize function to set con and icon array members */

Initialize(amesys, states);

/* It is possible that a submodel has changed some parameters. */
CheckSimParams(&tol1, &tol2, &hmax);

/* do not allow AmeReadFile to be interrupted by handler */
canHandlerLeave = 0;

/* Open file for results (in case of continuation run, t is overwritten
  by the time of the last step). */
AmeReadFile(&t, states);

/* allow signal handler to call ameexit */
canHandlerLeave = 1;

/* if signal handler has been called */
if (shouldMainThreadLeave)
{
  /* exit */
  AmeExit(2);
}

/* Set tolerance arrays. */
for(i=0;i<NUMSTATES;i++)
{

atol[i] = tol1;
rtol[i] = tol2;
}

/* Read linear analysis specification and locked variables status. */

SetLADetails(GetLAFileName(), NUMSTATES, NUMVARS, t, tol2, tol1, final_time-t);
SetUpLockedStates(GetCircuitName(), NUMSTATES);

if (IsAssemblyNecessary_())
{
    double dot[NUMSTATES];
    int local_flag;

    /* Perform the assembly. */

    consflag = 1;
    local_flag = 0;
    FunctionEval(dot, states, t, &local_flag);
    first_call = 1;

    consflag = 2;
    local_flag = 0;
    FunctionEval(dot, states, t, &local_flag);
    first_call = 1;
consflag = 0;

if(isstabrun_())
{
    fixed_time = t;
    success = Integrate(NUMSTATES, t, final_time, tinc, states, yh, acor, ewt, savf,
                        rstrtp, hmax, mode, fixed, rtol, atol,
                        stabilooption, iwrite, minimaldiscont);
}

if(isdynrun_())
{
    SetConRun(); /* Force Continuation run */
}
else
{
    /* Simulation complete. */
    AmeExit(!success);
}

/* Perform the integration. */
if(!FixedStep)
{
    success = Integrate(NUMSTATES, t, final_time, tinc, states, yh, acor,
                        ewt, savf, rstrtp, hmax, mode, fixed, rtol,
                        atol, stabilooption, iwrite, minimaldiscont);
}
else
{
    success = FixedStepIntegrate(NUMSTATES, t, final_time, tinc, states, yh,
                                  fixed_type, fixed_order, fixed_h);
}

/* we are about to exit, no need to exit from signal handler */
canHandlerLeave = 0;

/* Do a final call to WriteErrorCount closing state errors file. */
WriteErrorCount(1);

/* Simulation complete.*/
AmeExit(!success);
return 0;
**Circuit name**

First_cut2
Circuit schematics

Component submodels

control01 [JUN3M-1]

comparison junction differencing inputs

elementaryhydraulicprops [FP04-1]

indexed hydraulic fluid properties

<table>
<thead>
<tr>
<th>FP04-1 : Real parameters with non default values</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>density</td>
<td>kg/m**3</td>
<td>871</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FP04-1 : Parameter or variable group(s) containing parameter(s) with non default value</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>General properties</td>
<td></td>
</tr>
</tbody>
</table>
hsv_2pos3port_02 [HSV23_02-1]
2 position 3 port hydraulic valve

hydrnode3 [H3NODE3-1]
hydraulic junction 3 ports (pressure fixed by port 3)

motor02 [MO001-1]
ideal fixed displacement hydraulic motor

<table>
<thead>
<tr>
<th>MO001-1 : Real parameters with non default values</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>motor displacement</td>
<td>cc/rev</td>
<td>50</td>
</tr>
<tr>
<td>typical speed of motor</td>
<td>rev/min</td>
<td>1800</td>
</tr>
</tbody>
</table>

pmover01 [PM000-1]
constant speed prime mover

pmover01v [PMV00-1]
conversion of signal to a rotary speed in rev/min

powersensor_rotary [PTR00-1]
power/energy/activity sensor with offset and gain

powersensor_rotary_1 [PTR10-1]
power/energy/activity sensor with offset and gain

presscontrol01 [RV00-1]
simple hydraulic relief valve

pump01 [PU001C-1]
ideal fixed displacement hydraulic pump with volumetric and mechanical efficiency

<table>
<thead>
<tr>
<th>PU001C-1 : Real parameters with non default values</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pump displacement</td>
<td>cc/rev</td>
<td>75</td>
</tr>
<tr>
<td>typical speed of pump</td>
<td>rev/min</td>
<td>1800</td>
</tr>
<tr>
<td>if option=1, constant mechanical efficiency</td>
<td>null</td>
<td>0.9</td>
</tr>
<tr>
<td>if option=1, constant volumetric efficiency</td>
<td>null</td>
<td>0.9</td>
</tr>
</tbody>
</table>

signal03 [UD00-1]

piecewise linear signal source

<table>
<thead>
<tr>
<th>UD00-1 : Real parameters with non default values</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>output at start of stage 1</td>
<td>null</td>
<td>1</td>
</tr>
<tr>
<td>output at end of stage 1</td>
<td>null</td>
<td>1</td>
</tr>
</tbody>
</table>

signal03_1 [UD00-2]

piecewise linear signal source

<table>
<thead>
<tr>
<th>UD00-2 : Real parameters with non default values</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>output at start of stage 1</td>
<td>null</td>
<td>1800</td>
</tr>
<tr>
<td>output at end of stage 1</td>
<td>null</td>
<td>1800</td>
</tr>
</tbody>
</table>

signalsink [SSINK-1]

plug for signal port

tank01 [TK000-2]

tank modelled as constant pressure source

tank01_2 [TK000-3]

tank modelled as constant pressure source
tank01_3 [TK000-4]
tank modelled as constant pressure source

**tank01_4 [TK000-5]**
tank modelled as constant pressure source

**Line submodels**

**h2port [HL03-1]**
compressibility + friction hydraulic line (C-R-C)

<table>
<thead>
<tr>
<th>HL03-1 : External variables with non default startvalues</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pressure at port 1</td>
<td>bar</td>
<td>150</td>
</tr>
<tr>
<td>pressure at port 2</td>
<td>bar</td>
<td>150</td>
</tr>
</tbody>
</table>

**h2port_3 [HL000-1]**
simple compressibility hydraulic line (C)

<table>
<thead>
<tr>
<th>HL000-1 : External variables with non default startvalues</th>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pressure at port 1</td>
<td>bar</td>
<td>150</td>
</tr>
</tbody>
</table>

**Run Parameters**

<table>
<thead>
<tr>
<th>Run parameter</th>
<th>Value</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run type</td>
<td>Single</td>
<td></td>
</tr>
<tr>
<td>Integrator type</td>
<td>Standard</td>
<td></td>
</tr>
<tr>
<td>Start time</td>
<td>0</td>
<td>s</td>
</tr>
<tr>
<td>End time</td>
<td>10</td>
<td>s</td>
</tr>
<tr>
<td>Communication interval</td>
<td>0.1</td>
<td>s</td>
</tr>
<tr>
<td>Tolerance</td>
<td>1e-005</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>Maximum time step</td>
<td>1e+030  s</td>
<td></td>
</tr>
<tr>
<td>Solver type</td>
<td>Regular</td>
<td></td>
</tr>
<tr>
<td>Error type</td>
<td>Relative</td>
<td></td>
</tr>
<tr>
<td>Simulation mode</td>
<td>Stabilizing + Dynamic</td>
<td></td>
</tr>
<tr>
<td>Lock non propagating states</td>
<td>On</td>
<td></td>
</tr>
<tr>
<td>Discontinuity printout</td>
<td>Off</td>
<td></td>
</tr>
<tr>
<td>Activity index calculation</td>
<td>On</td>
<td></td>
</tr>
<tr>
<td>Holds input constant</td>
<td>Off</td>
<td></td>
</tr>
</tbody>
</table>

**LA status**

**Free states**

<table>
<thead>
<tr>
<th>Submodel</th>
<th>Variable</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 h2port [HL03-1]</td>
<td>pressure at port 1</td>
<td>bar</td>
</tr>
<tr>
<td>~ pump01 [PU001C-1]</td>
<td>pressure at port 2</td>
<td>bar</td>
</tr>
<tr>
<td>2 h2port_3 [HL000-1]</td>
<td>pressure at port 1</td>
<td>bar</td>
</tr>
<tr>
<td>~ hsv_2pos3port_02 [HSV23_02-1]</td>
<td>pressure at port A</td>
<td>bar</td>
</tr>
<tr>
<td>3 hsv_2pos3port_02 [HSV23_02-1]</td>
<td>fractional spool position</td>
<td></td>
</tr>
<tr>
<td>4 hsv_2pos3port_02 [HSV23_02-1]</td>
<td>fractional spool velocity</td>
<td>1/s</td>
</tr>
<tr>
<td>5 h2port [HL03-1]</td>
<td>pressure at port 2</td>
<td>bar</td>
</tr>
<tr>
<td>~ hydrnode3 [H3NODE3-1]</td>
<td>junction pressure</td>
<td>bar</td>
</tr>
</tbody>
</table>

**Observer variables**
<table>
<thead>
<tr>
<th>Submodel</th>
<th>Variable</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 h2port [HL03-1]</td>
<td>pressure at port 1</td>
<td>bar</td>
</tr>
<tr>
<td>~ pump01 [PU001C-1]</td>
<td>pressure at port 2</td>
<td>bar</td>
</tr>
<tr>
<td>2 h2port_3 [HL000-1]</td>
<td>pressure at port 1</td>
<td>bar</td>
</tr>
<tr>
<td>~ hsv_2pos3port_02 [HSV23_02-1]</td>
<td>pressure at port A</td>
<td>bar</td>
</tr>
<tr>
<td>3 hsv_2pos3port_02 [HSV23_02-1]</td>
<td>fractional spool position</td>
<td></td>
</tr>
<tr>
<td>4 hsv_2pos3port_02 [HSV23_02-1]</td>
<td>fractional spool velocity</td>
<td>1/s</td>
</tr>
<tr>
<td>5 h2port [HL03-1]</td>
<td>pressure at port 2</td>
<td>bar</td>
</tr>
<tr>
<td>~ hydnode3 [H3NODE3-1]</td>
<td>junction pressure</td>
<td>bar</td>
</tr>
</tbody>
</table>

**Appendix D: AMESIM Graphs**

![Figure 37: AMESim Circuit](image)
Figure 38: Relief Valve Flow Rate (min: 10.3028 LPM) (max: 81 LPM)

Figure 39: Tank Flow (min: -56.6785 LPM)
Figure 40: Motor Pressure (149.526 bar)

Figure 41: Motor Flow (max: 70.6972 LPM) (min: 56.6785 LPM)
Figure 42: Positive Power (14130.2 W)

Figure 43: Negative Power (37.52 W)
Figure 44: Motor Torque (74.96 Nm)

Figure 45: Net Power Consumption (max: 25137.4 W) (min: 10946.1 W)
Figure 46: Power Response