7

Bond Graph Modeling

Preview

When I read a technical paper written by an author from whom I never have read anything before, I usually find that I have the biggest problems not with mastering the intellectual challenges that the paper presents, but with understanding the author's terminology, and relating it back to concepts that I am familiar with. In this context, modeling is a notoriously difficult subject since it is so utterly interdisciplinary. It is therefore one of the major goals of this text to introduce a variety of different modeling concepts (terminologies), and relate them to each other. In this chapter, we shall discuss several graphical modeling techniques among which, in more detail, the *bond graph modeling technique* which has found wide–spread acceptance and utilization among a number of modelers from several application areas, but in particular among mechanical engineers. We shall see that bond graphs are very easy to relate back to previously introduced concepts, and yet, they are somewhat difficult to read on a first glance.

7.1 Block Diagrams

We have used block diagrams previously in this text on an *ad hoc* basis without properly introducing them. I felt that this approach was quite adequate since the interpretation of a block diagram is straightforward. But let me now go back, and introduce block diagrams formally.

Block diagrams consist of *blocks* that are connected by *paths*. The paths represent signals, and the blocks are transducers that transform one (set of) signal(s) into another. While these two types of modeling elements would, in theory, suffice to draw any block diagram, two frequently used types of transducers are usually being represented by special symbols: the *take–off point*, and the *summer*. Fig.7.1 shows the four elementary block diagram modeling types.

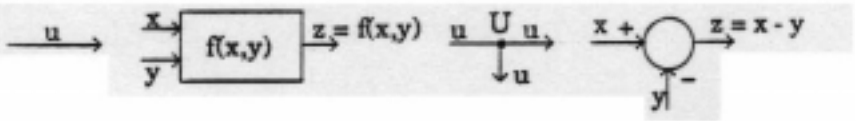

Figure 7.1. Modeling elements of a block diagram

Let us return once more to our simple electrical circuit of Fig.7.2:

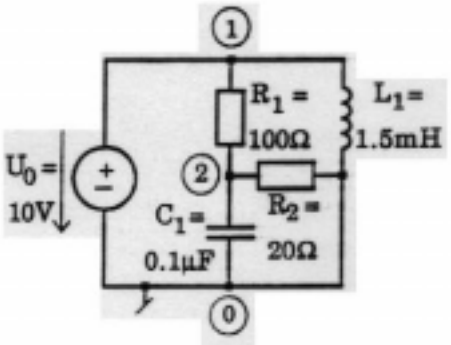

Figure 7.2. Simple passive circuit

We wish to construct a block diagram for it. We start with a set of equations that describes the circuit correctly in terms of a *computational structure*. One such set of equations had been derived in Chapter 3. Let us write down these equations here once more.

$$i_1 = u_1/R_1 \tag{7.1a}$$

$$i_2 = u_2/R_2 \tag{7.1b}$$

$$\frac{di_L}{dt} = u_L/L_1 \tag{7.1c}$$

$$\frac{du_C}{dt} = i_C/C_1 \tag{7.1d}$$

$$u_1 = U_0 - u_C \tag{7.1e}$$
$$u_2 = u_C \tag{7.1f}$$
$$u_L = u_1 + u_2 \tag{7.1g}$$
$$i_0 = i_1 + i_L \tag{7.1h}$$
$$i_C = i_1 - i_2 \tag{7.1i}$$

When constructing a block diagram, I always start by drawing all integrators as boxes (blocks) below each other. The inputs to these boxes are currently *unresolved variables*. For each unresolved variable, I simply proceed to the equation that defines that variable, and draw this equation into the block diagram, until all unresolved variables have been reduced to true inputs of the system. Each equation that has been used in the block diagram is removed from the set of unused equations. In many cases, no unused equations are left at the end. If equations have not been used, these are *output equations* that can simply be added to the block diagram in the end.

This algorithm leads to the block diagram shown in Fig.7.3:

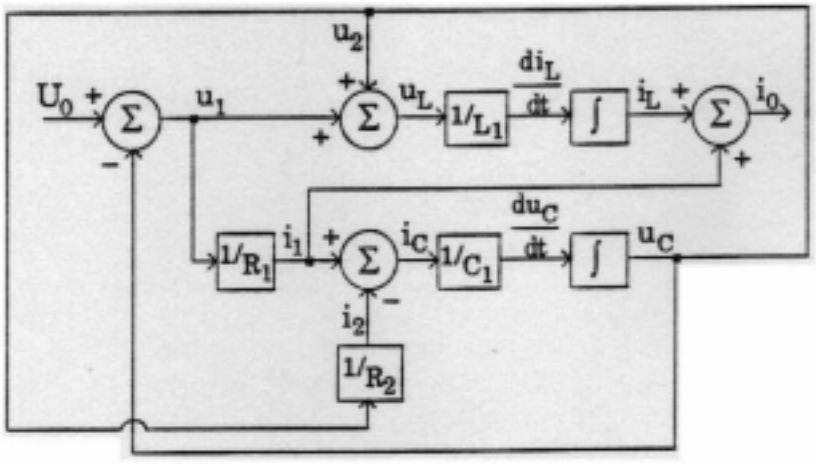

Figure 7.3. Block diagram of the passive circuit

which can thereafter be simplified as shown in Fig.7.4:

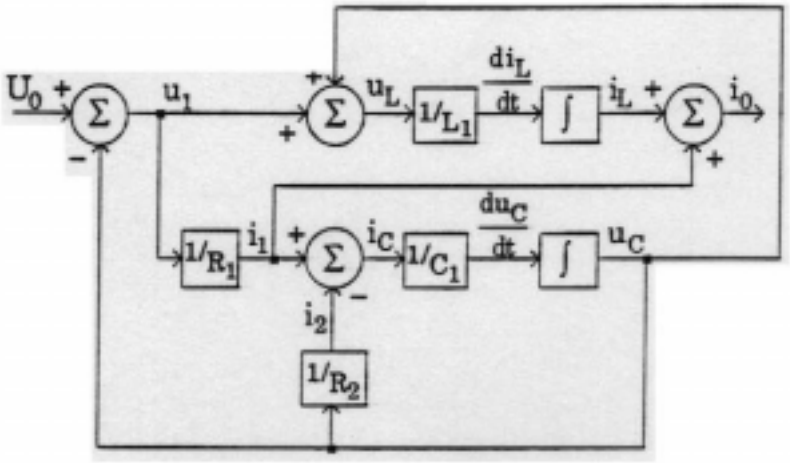

Figure 7.4. Simplified block diagram of the passive circuit

While the block diagram clearly shows the *computational structure* of the system, it does not preserve any of the topological properties of the system. As we have seen in the past, a small change in the circuit may force us to rearrange the computational structure entirely, and therefore, its corresponding block diagram may bear little resemblance with the previously used one.

This is a clear disadvantage of block diagrams, and therefore, block diagrams are not commonly used to describe electrical circuits. They are mostly used by *control engineers* because control engineers *force* their circuits to behave in such a way that the computational structure and the topological structure coincide. This can be achieved by placing a *voltage follower circuit* between any two consecutive blocks of the block diagram as shown in Fig.7.5:

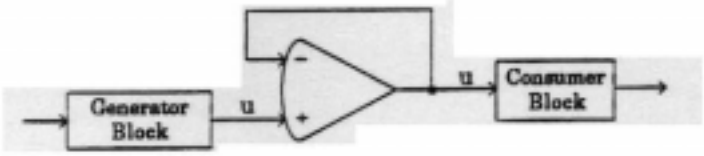

Figure 7.5. Impedance decoupling with a voltage follower

The voltage follower circuit decouples the two consecutive blocks. The *generator block* produces a control signal u that is independent of the *consumer block*, i.e., the voltage produced by the generator does not depend on the current that is drawn by the consumer. While this technique is common practice among control engineers, the voltage follower circuits are never shown in the block diagrams. They are simply assumed to be there.

However, if we connect two wires in an arbitrary electrical circuit, we actually connect *two* variables at the same time, namely one *across variable*, the potential v, and one *through variables*, the current i. In the block diagram, these two variables get separated from each other, and it is this fact which destroys the symmetry between the *topological structure* and the *computational structure*.

7.2 Signal Flow Graphs

Another frequently used graphical modeling tool is the signal flow graph. A strong relation exists between block diagrams and signal flow graphs. Fig.7.6 shows the correspondence between the elementary modeling tools in a block diagram (top) and the equivalent elementary modeling tools in a signal flow graph (bottom).

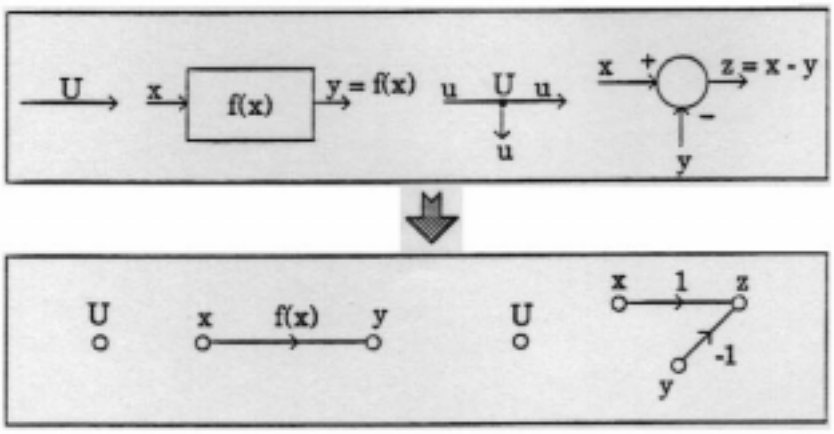

Figure 7.6. Elements of block diagrams *vs* signal flow graphs

Each "path" in a block diagram turns into a "node" in the signal flow graph. Each "box" in the block diagram becomes a "path" in the signal flow graph. In this respect, the two diagrams could be called *dual representations*. However, no signal flow graph equivalent exists for a multi–port block diagram box. In this respect, signal flow graphs are a little less powerful than block diagrams.

When converting a block diagram into a signal flow graph, I always proceed along the following lines. First, I assign a name to all signals (paths) in the block diagram that have not yet been named. I usually call these signals e_1, e_2, etc. In our example, all signals have been named already. Next, I draw nodes (little circles) where ever the block diagram had a signal path. Finally, I connect these nodes with paths according to the correspondence shown in Fig.7.6.

For our passive circuit, we find the signal flow graph shown in Fig.7.7:

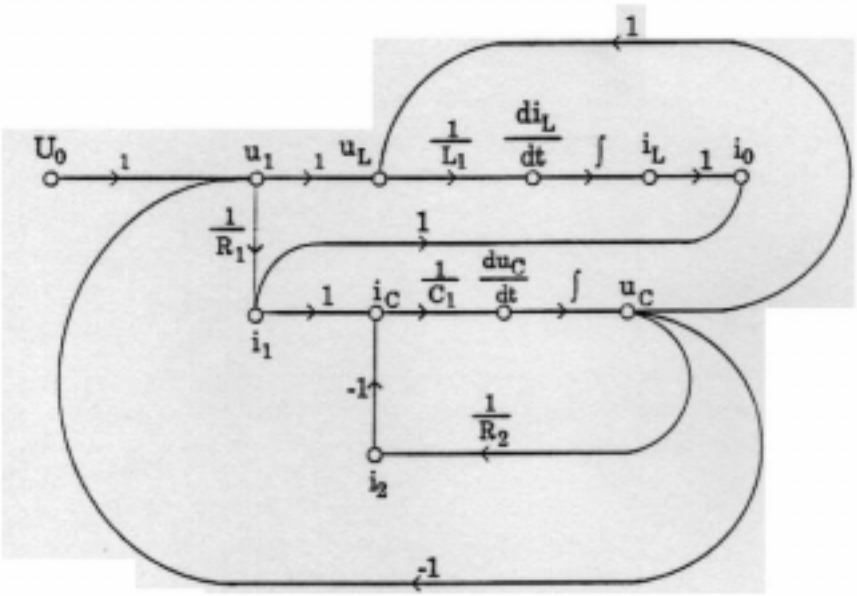

Figure 7.7. Signal flow graph of the passive circuit

The signal flow graph can be simplified by eliminating all nodes that represent *series connections* of paths. The corresponding path functions are simply multiplied with each other. The result of this simplification is shown in Fig.7.8.

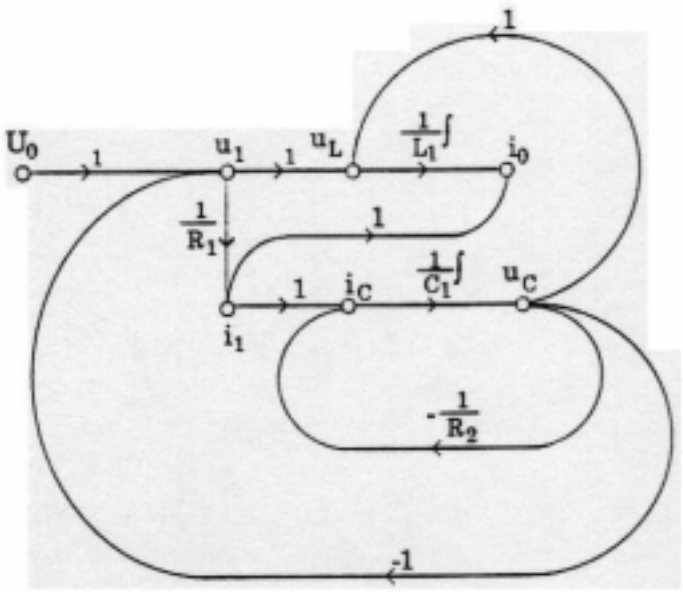

Figure 7.8. Simplified signal flow graph of the passive circuit

When interpreting a signal flow graph, it is essential to realize that nodes represent both take–off points and summers at the same time. Incoming paths represent summing functions, while outgoing paths represent take–offs. Looking at the node u_1, for example, we find that:

$$u_1 = 1 * U_0 + (-1) * u_C \tag{7.2}$$

and *not*:

$$u_1 = 1 * U_0 + (-1) * u_C - 1 * u_L - \frac{1}{R_1} * i_1 \tag{7.3}$$

Once this small detail has been understood, signal flow graphs become as easy to read as block diagrams, and they really are more or less equivalent to each other. In particular, they share the same advantages and disadvantages, namely they capture the *computational structure* while they do not preserve the *topological structure* of the system which they represent. They are about equally frequently found in control engineering texts. Some texts operate on signal

flow graphs only while others use block diagrams exclusively. The most prominent use of signal flow graphs is for the determination of a transfer function between any two nodes (signals) in a linear circuit using *Mason's rule*. I don't care too much for signal flow graphs since they are not "computer–friendly", i.e., I have a hard time drawing them neatly on my MacIntosh. Therefore, I won't use them much in this text.

7.3 Power Bonds

While block diagrams and signal flow graphs preserve the computational structure of a system only, circuit diagrams reflect the topological structure exclusively. Moreover, they are restricted to use in electrical systems. For these reasons, H.M. Paynter, a professor at M.I.T., recognized around 1960 the need for yet another graphical representation of systems which would be able to show simultaneously the topological as well as the computational structure, and which would be general, i.e., could be applied to all kinds of physical systems. He came up with the bond graph [7.14]. A bond, represented by a bold harpoon, is nothing but a connector that simultaneously connects two variables, one across variable, in bond graph terminology usually referred to as the "effort" e, and one through variable, called the "flow" f. The bond is shown in Fig.7.9:

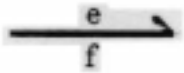

Figure 7.9. The bond

The bond graph literature is not systematic with respect to the bond graph conventions. The harpoon is sometimes shown to the left and sometimes to the right of the bond, and the effort variable is sometimes indicated on the side of the harpoon and sometimes away from the harpoon. This inconsistency can be explained by the fact that most bond graphers viewed the bond graph methodology as a pure modeling aid to be used with paper and pencil. The fact that

a model represents always a *codified* form of knowledge occurred to them at best as an afterthought.

However, if we wish to use bond graphs as a tool to formalize a model and to formulate it as input to a computer program (as we shall do in this chapter), we need to be more rigorous. For this purpose, I decided that the harpoon must sit always on the left of the bond, and the effort variable is always indicated on the side of the harpoon, while the flow variable is indicated on the side away from the harpoon.

The bond is able to preserve the topological structure because the two types of variables are not dislocated from each other. Bonds connect either to system elements, such as a resistor R which, in bond graph terminology, is a *single port element* (since both variables u_R and i_R are connected simultaneously), or they connect to other bonds in a *junction*. Two different types of junctions exist, the so–called "0–junction", and the so–called "1–junction" [7.11]. In a 0–junction, all effort variables are equal, while all flow variables add up to zero. A 0–junction is thus equivalent to a node in an electric circuit diagram, or a *node* in a DYMOLA [7.7] program. In a 1–junction, all flow variables are equal, while all effort variables add up to zero. The two junction types are shown in Fig.7.10.

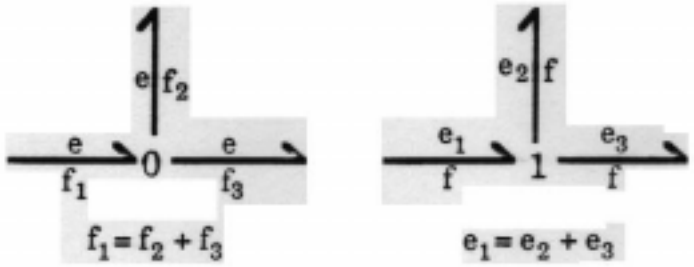

Figure 7.10. The two junction types

The 0–junction thus represents Kirchhoff's current law, while the 1–junction represents Kirchhoff's voltage law. If a bond connects two junctions, one will always be of the 0–junction type, while the other is of the 1–junction type, i.e., in a bond graph, 0–junctions and 1–junctions toggle among each other. Neighboring junctions of the same gender can be combined into one.

7.4 Bond Graphs for Electrical Circuits

Let me explain by means of my passive circuit, how a bond graph can be constructed. Electric circuit designers have the habit to choose one node as their *reference node*, usually the ground. Since the ground appears at many places in a complex circuit, they usually do not bother to connect all the grounds together in a circuit diagram. The circuit diagram shown in Fig.7.11 is equivalent to the previously shown diagram of Fig.7.2.

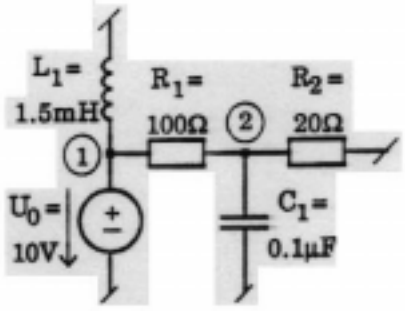

Figure 7.11. Electric circuit diagram of the passive circuit

In the bond graph, we shall do the same thing. We start by representing each circuit node by a 0–junction except for the reference node which is drawn like in a circuit diagram. We then represent each branch of the circuit diagram by a pair of bonds connecting two 0–junctions with a 1–junction between them. We let the harpoons point in the same direction that we picked for the branch currents. Finally, we attach the circuit elements to the 1–junctions with the harpoons directed away from the junction for passive circuit elements, and directed towards the junction for sources [7.1]. This algorithm leads to the bond graph shown in Fig.7.12:

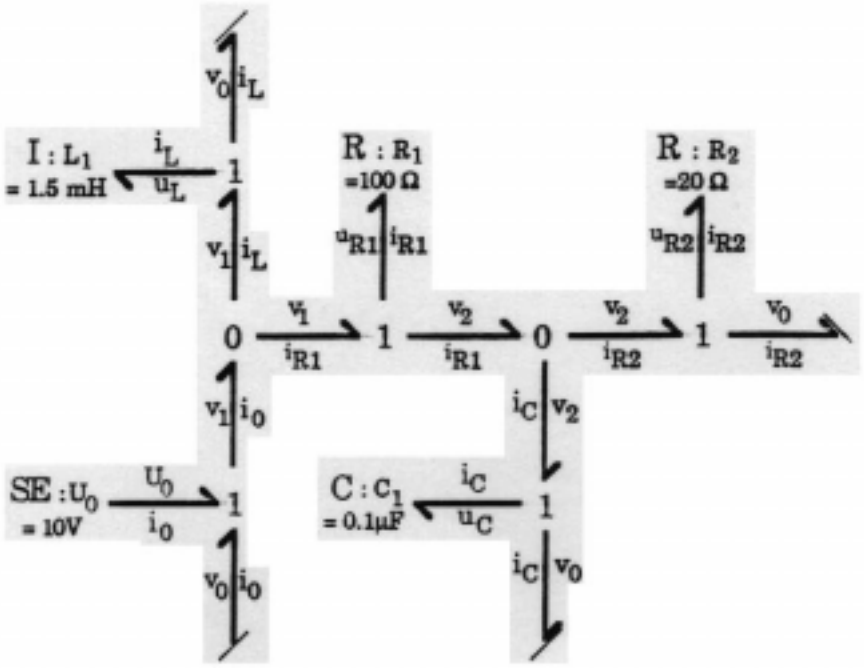

Figure 7.12. Bond graph of the passive circuit

This bond graph can still be simplified quite a bit. Remember that, when writing down the circuit equations, we always skipped Kirchhoff's current law for the reference node. In DYMOLA [7.7] notation, we placed a "." in the current's position of the *Common* submodel's cut which instructs the DYMOLA compiler to skip the current equation. We did this because we knew that this equation would be redundant. Accordingly, since the reference node in the bond graph actually represents a degenerated 0–junction which represents Kirchhoff's current law for the reference node, we can as well leave out the reference node together with all the bonds connecting to it. This rule makes also physical sense. The power that flows through bonds connecting to the reference node is the product of the current flowing through the bond and the potential of the reference node. Since this potential can, without loss of generality, be assumed to be zero, we can say that no power flows into or out of the reference node. Thus, it makes physical sense to eliminate such bonds from the bond graph.

Also, if a junction has only two bonds attached to it, and if their harpoons point both in the same direction, we can eliminate this junction, and amalgamate the two bonds into one [7.1]. This procedure leads to the simplified bond graph of Fig.7.13:

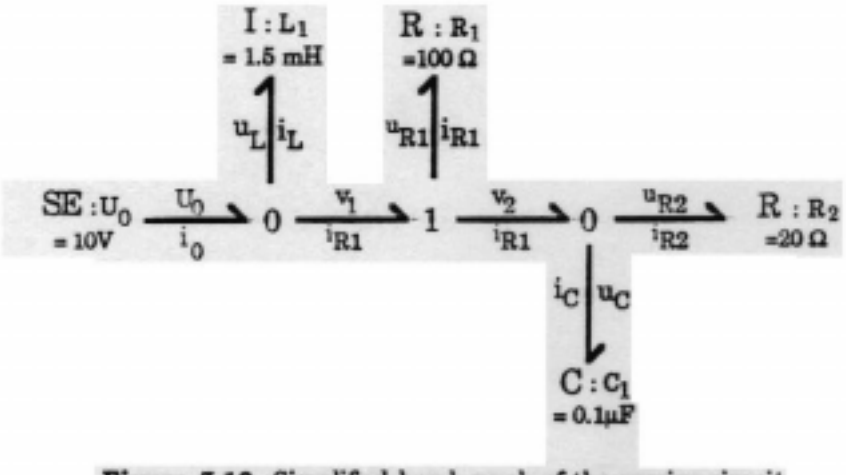

Figure 7.13. Simplified bond graph of the passive circuit

Notice that the bold printed element type denotes the type of equation that describes the circuit element. **R** stands for resistance, **C** stands for capacitance (or compliance), **I** stands for inductance (or inertia), and **SE** stands for effort source. Correspondingly, **SF** stands for flow source.

It is quite evident that the bond graph preserves the topological structure. However, we have not seen yet how it represents the computational structure at the same time. For this purpose, we introduce the notion of *bond graph causality* [7.11].

Each bond is involved in two equations, one to determine its effort e, and the other to determine its flow f. Each of these two equations is formulated at one of the two ends of the bond. The causality is indicated by a short stroke perpendicular to the bond which is placed at one of the two ends of the bond. It marks the side of the bond at which the *flow* variable is being determined.

Let me explain this concept by means of a simple example. If a bond connected to a resistance **R** has its stroke at the end at which **R** is attached to the bond, as shown in Fig.7.14a,

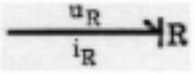

Figure 7.14a. Causal bond connected to a resistance

then we need to compute the flow variable at the resistance which leads to the equation:

$$i_R = u_R/R \tag{7.4}$$

This means that u_R must be computed elsewhere, namely at the other end of the bond. However, if the causality is assigned the other way around, such as shown in Fig.7.14b:

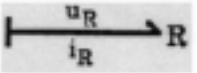

Figure 7.14b. Causal bond connected to a resistance

then, the effort needs to be computed at the resistance, and we obtain the equation:

$$u_R = R \cdot i_R \tag{7.5}$$

whereas the equation used to compute the flow will be formulated at the other end of the bond.

For a resistance, both causalities are physically and computationally meaningful. However, for effort sources and flow sources, the causality is physically determined as shown in Fig.7.15,

Figure 7.15. Necessary causality for effort and flow sources

whereas for capacitances and inductances, the desired causality is dictated by computational requirements since we wish to numerically

integrate over all state variables rather than to *differentiate* them. The desired causalities for these elements are shown in Fig.7.16.

Figure 7.16. Desired causality for capacitances and inductances

Finally, rules can be specified for the two junction types. Since only one flow equation can be specified for every 0–junction, only one of the bonds can compute the flow at the junction, i.e., exactly one stroke must be located at every 0–junction. Since only one effort equation can be formulated for every 1–junction, only one of the bonds can compute the effort, i.e., only one stroke can be located away from every 1–junction. Fig.7.17 shows the causal bond graph for our passive circuit.

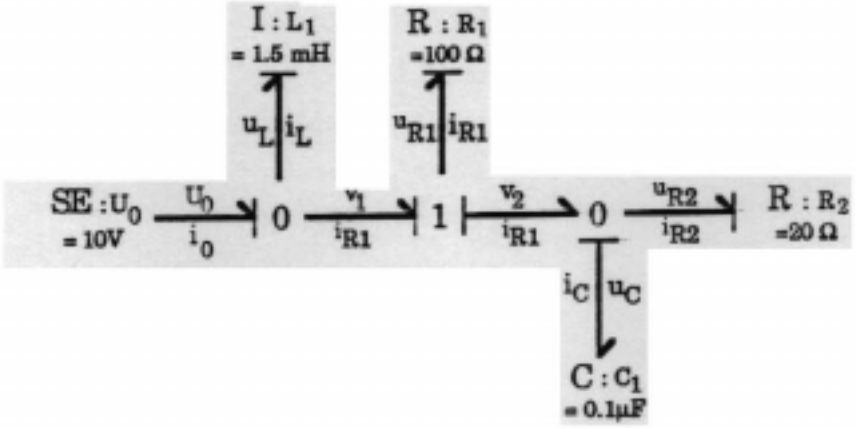

Figure 7.17. Causal bond graph of the passive circuit

In this example, all conditions can be satisfied, and the solution is unique. This is the preferred situation. If not all necessary conditions can be satisfied, we are confronted with a *non–causal system*. This case occurs for example if we try to parallel connect two voltage sources with different voltage values. If we cannot satisfy all desired conditions, i.e., if we run into wrong causalities at either C or I

elements, we are confronted with a *degenerate system*, i.e., the true system order is lower than the number of integrators would indicate. We are thus confronted with a *structural singularity*. If we have a choice in assigning the causalities without offending any of the rules, the model contains an *algebraic loop*.

Please, notice that the specification of the voltages as "efforts" and the currents as "flows" is somewhat arbitrary. The bond graph is completely symmetrical in this respect. Had we decided to call the currents "efforts" and the voltages "flows", then all junctions would change their gender, and the causality laws for capacitances, inductances, and the two types of sources would be reversed. We call this the *dual bond graph*. It will be discussed later in this chapter in more detail.

7.5 Bond Graphs for Mechanical Systems

If all we could do with bond graphs were to have yet another tool to describe electrical circuits, this would not be very exciting since we already had a topological description mechanism (the circuit diagram) which comes more natural to electrical engineers than bond graphs. However, the concepts of *effort* and *flow* are much more general than that.

If we take three levers, and join them in one point, we notice that the *velocities* of the three levers at that point must be equal, while the *forces* add up to zero. In simple analogy, we are therefore inclined to come up with the following equivalencing scheme:

$$\begin{array}{lcccl} effort & \Longleftrightarrow & potential & \Longleftrightarrow & velocity \\ flow & \Longleftrightarrow & current & \Longleftrightarrow & force \end{array}$$

Once we made this decision, all other quantities in the bond graph are determined. Let us look at Newton's law:

$$m \cdot \frac{dv}{dt} = \sum_{\forall i} f_i \quad \Rightarrow \quad \frac{dv}{dt} = \frac{1}{m} \cdot \sum_{\forall i} f_i \quad \Rightarrow \quad v = \frac{1}{m} \cdot \int_0^t \sum_{\forall i} f_i d\tau + v_0 \quad (7.6)$$

We immediately notice the similarity with the voltage/current relationship for a capacitor:

$$C \cdot \frac{du_C}{dt} = i_C \quad \Rightarrow \quad \frac{du_C}{dt} = \frac{1}{C} \cdot i_C \quad \Rightarrow \quad u_C = \frac{1}{C} \cdot \int_0^t i_C d\tau + u_{C_0} \tag{7.7}$$

Thus, the mechanical *mass* corresponds to an electrical *capacitor*. Let us now look at a spring. For the spring, we have the relationship:

$$f_{S_p} = k \cdot x \quad \Rightarrow \quad \frac{df_{S_p}}{dt} = k \cdot v \tag{7.8}$$

which can immediately be compared to the voltage/current relationship for an inductor:

$$\frac{di_L}{dt} = \frac{1}{L} \cdot u_L \tag{7.9}$$

Thus, the inductance can be equivalenced to the inverse of the spring constant which is sometimes called the *compliance* of the spring. Finally, we can look at friction phenomena:

$$f_{F_r} = b \cdot v \quad \Rightarrow \quad v = \frac{1}{b} \cdot f_{F_r} \tag{7.10}$$

which can be compared to Ohm's law for resistors, i.e., the electric conductance can be compared to the friction constant b.

Unfortunately, "bond graphers" around the world did it just the other way around. They once decided to let the forces be called efforts, and the velocities be called flows. As I explained before, due to symmetry, both assumptions are equally acceptable. I assume that the original reason for this decision was related to a mixup of the terms "effort" and "flow" with the terms "cause" and "effect". In many mechanical systems, the forces are considered the "causes", and the velocities are considered their "effects" (except if I drive my car against a tree). In any event, the decision is an arbitrary one, and, in order to be in agreement with the existing literature on bond graphs, I shall bow to the customary convention.

In this case, of course, we now need to compare Newton's law to the inductor rather than the capacitor:

$$L \cdot \frac{di_L}{dt} = u_L \quad \Rightarrow \quad \frac{di_L}{dt} = \frac{1}{L} \cdot u_L \quad \Rightarrow \quad i_L = \frac{1}{L} \cdot \int_0^t u_L d\tau + i_{L_0} \tag{7.11}$$

Thus, the mechanical *mass* corresponds now to an electrical *inductor*. In accordance with this convention, the spring compares to the capacitor as shown below:

$$\frac{du_C}{dt} = \frac{1}{C} \cdot i_C \tag{7.12}$$

i.e., the *capacitor* corresponds to the *compliance* of the spring. Finally, the friction constant b corresponds now to the electrical resistance, rather than to the electrical conductance.

Notice that the terms "effort" and "flow" were defined by researchers dealing with bond graphs. On the other hand, the terms "across" variable and "through" variable were defined by other researchers who probably weren't even aware of the ongoing research on bond graphs. These researchers defined the forces as "across" variables and the velocities as "through" variables (which makes sense). For this reason, the correspondence of "effort" with "across" variable, and "flow" with "through" variable is not consistent throughout the literature. However, in this text, I shall use "effort" and "across" variable interchangeably. For this purpose, I simply *redefined* the term "across" variable to mean "effort". Thus, my mechanical "across" variables are the forces and torques, whereas my mechanical "through" variables are the velocities and angular velocities. As I mentioned earlier, this assignment is purely arbitrary. It simply made sense to use a consistent terminology at least within this text, bearing the risk of a potential confusion for readers who are familiar with the more conventional definitions of "across" and "through" variables.

Let us go ahead and derive a bond graph for the simple mechanical system that had previously been presented in Fig.4.2. To refresh our memory, here it is once more.

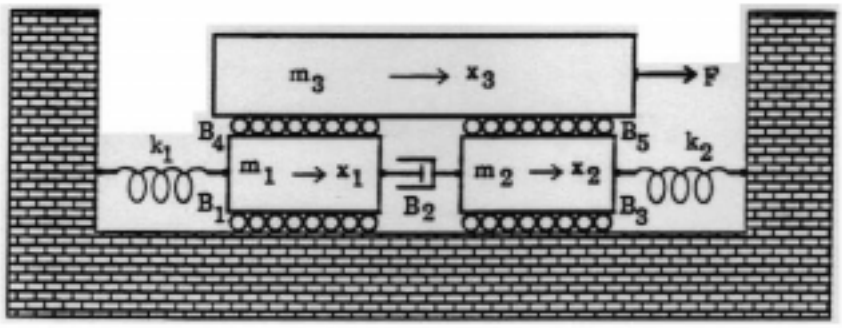

Figure 7.18. Simple translational problem

We start by identifying all free moving bodies. We place 1–junctions for each of their velocities. Where ever two bodies interact with each other, we connect their junctions with branches consisting of two bonds and one 0–junction in between, and attach all interacting elements to that 0–junction. Newton's law (or rather the d'Alembert principle) is formulated at the 1–junctions themselves [7.1]. Fig.7.19 shows the bond graph for the simple translational system of Fig.7.18.

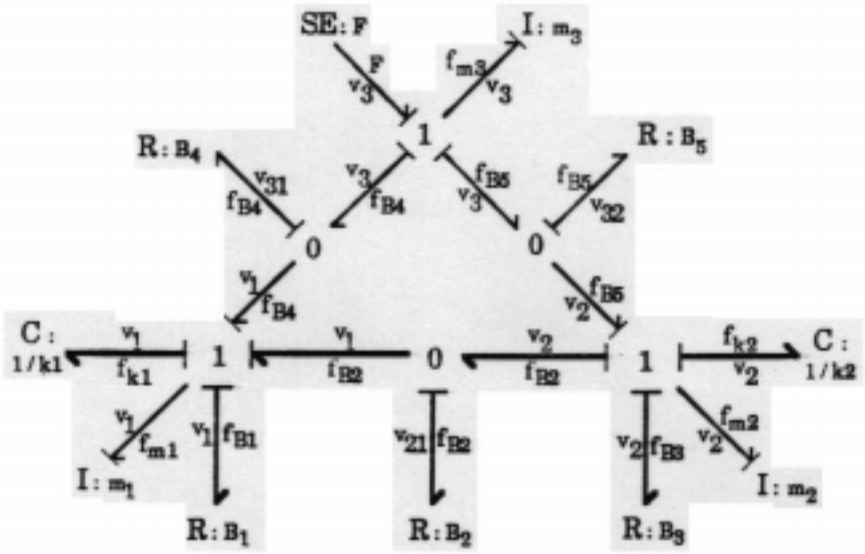

Figure 7.19. Bond graph for the translational problem

Rotational systems work exactly the same way. Here, the *torques* are taken for the "effort" variables, and the *angular velocities* are taken for the "flows".

7.6 Generalization to Other Types of Systems

Besides the two basic quantities effort e and flow f, we often make use of two additional derived quantities, namely the *generalized momentum*:

$$p = \int_0^t e \, d\tau \tag{7.13}$$

and the *generalized displacement*:

$$q = \int_0^t f \, d\tau \tag{7.14}$$

In electrical systems, the generalized momentum is the *flux* through a coil, and the generalized displacement is the *charge* in a capacitor. In translational mechanical systems, these are the *momentum* and the *displacement* (bond graphs were invented by a mechanical engineer), and in rotational mechanical systems, they are the *angular momentum* and the *angular position*.

All these quantities are common to a large variety of other physical systems as well, as are the two Kirchhoff laws. Hydraulic, pneumatic, and acoustic systems operate similarly to the electrical and mechanical ones. In all these systems, the *pressure* is defined as the effort variable, while the *volume flow rate* is defined as the flow variable. The derived quantities are the *pressure momentum* and the *volume*.

The element laws, however, may look different for different types of systems. In particular, it may be noted that the equivalent to Ohm's law for these types of systems is often *non-linear*. For example, the relation between the pressure (effort) p and the flow q in a (turbulent) hydraulic valve is quadratic:

$$\Delta p \propto q^2 \tag{7.15}$$

Notice the confusing nomenclature. The symbols p and q are the most commonly used symbols in the hydraulic and pneumatic literature to denote pressures and flows. However, these are *effort* and *flow* variables, and not *generalized momentums* and *generalized displacements*.

Table 7.1 presents a summary of the four generic variables for the most commonly used physical system types.

Table 7.1. Power–(e,f) and energy–(p,q) variables [7.17]

	Effort	Flow	Generalised Momentum	Generalised Displacement
	e	f	p	q
Electrical	voltage u $[V]$	current i $[A]$	flux Φ $[V\ sec]$	charge q $[A\ sec]$
Translational	force F $[N]$	velocity v $[m\ sec^{-1}]$	momentum $\mathcal{I}$ $[N\ sec]$	displacement x $[m]$
Rotational	torque T $[N\ m]$	angular velocity ω $[rad\ sec^{-1}]$	twist $\mathcal{T}$ $[N\ m\ sec]$	angle ϕ $[rad]$
Hydraulic	pressure p $[N\ m^{-2}]$	volume flow q $[m^3\ sec^{-1}]$	pressure momentum Γ $[N\ m^{-2}\ sec]$	volume V $[m^3]$
Chemical	chemical potential μ $[J\ mole^{-1}]$	molar flow ν $[mole\ sec^{-1}]$	—	number of moles n $[mole]$
Thermo-dynamical	temperature T $[°K]$	entropy flow $\frac{dS}{dt}$ $[W\ °K^{-1}]$	—	entropy S $[J\ °K^{-1}]$

7.7 Energy Transducers

Until now, we have looked at different types of systems in isolation. However, one of the true strengths of the bond graph approach is the ease with which transitions from one form of system to another can be made, while ensuring that the energy (or power) conservation rules are satisfied.

The power in an electrical system can be expressed as the product of voltage and current, or in terms of the bond graph terminology:

$$P = e \cdot f \tag{7.16}$$

The power is measured in Watt $[W] = [V \cdot A]$. The energy is the integral of the power over time

$$E = \int_0^t e \cdot f \, d\tau \tag{7.17}$$

which is measured in Joule $[J] = [W \cdot sec] = [V \cdot A \cdot sec]$.

Until now, we assumed that we could select the effort and flow variables more or less freely. However, this assumption is incorrect. In all system types, the variables are chosen such that their product results in a variable of type *power* [7.17].

In an energy transducer (such as a transformer, or a DC–motor), the energy (or power) which is fed into the transducer is converted from one energy form to another, but it is never lost. Consequently, the energy that enters the transducer at one end must come out in one or more different form(s) at the other. A "loss–less" energy transducer may, for example, transform electrical energy into mechanical energy. In reality, every energy transducer "loses" some energy, but the energy does not really disappear — it is simply transformed into heat.

The above energy conservation law can be satisfied in exactly two ways in an "ideal" (i.e., loss–less) energy transducer. One such transducer is the *ideal transformer*. It is governed by the following set of relationships:

$$e_1 = m \cdot e_2 \tag{7.18a}$$
$$f_2 = m \cdot f_1 \tag{7.18b}$$

The ideal transformer is placed between two junctions. Two types of causalities are possible as shown in Fig.7.20.

Figure 7.20. Causality bond graph for the ideal transformer

Examples of transformers are the electrical transformer (shown in Fig.7.21a), the mechanical gear (shown in Fig.7.21b), and a mechano–hydraulic pump (shown in Fig.7.21c).

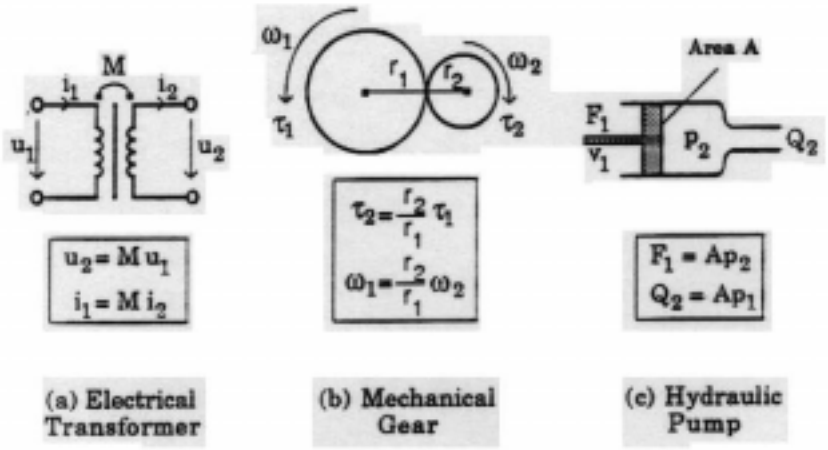

Figure 7.21. Examples of ideal transformers

The other type of energy transducer is the *ideal gyrator*. Its behavior is governed by the equations:

$$e_1 = r \cdot f_2 \qquad (7.19a)$$

$$e_2 = r \cdot f_1 \qquad (7.19b)$$

Also the ideal gyrator exhibits two forms of causalities as shown in Fig.7.22.

Figure 7.22. Causality bond graph for the ideal gyrator

Examples of gyrators are most electro–mechanical converters, for example the DC–motor shown in Fig.7.23.

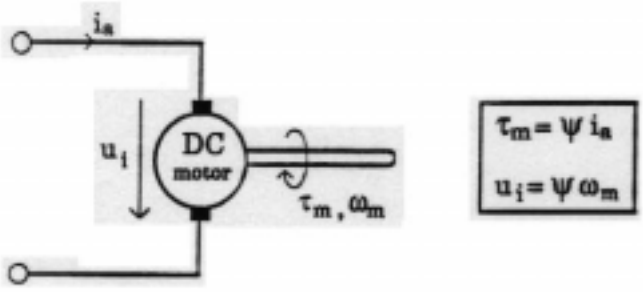

Figure 7.23. Example of an ideal gyrator

Notice that no real difference exists between the two transducer types [7.2,7.3]. If the effort and flow variables in the mechanical system were exchanged (as earlier suggested), the DC–motor would in fact become a transformer.

Let us go through an example. We want to model the mechanical system of Fig.4.5 driving it with an armature controlled DC–motor with constant field such as the one depicted in Fig.4.13. The resulting bond graph is shown in Fig.7.24.

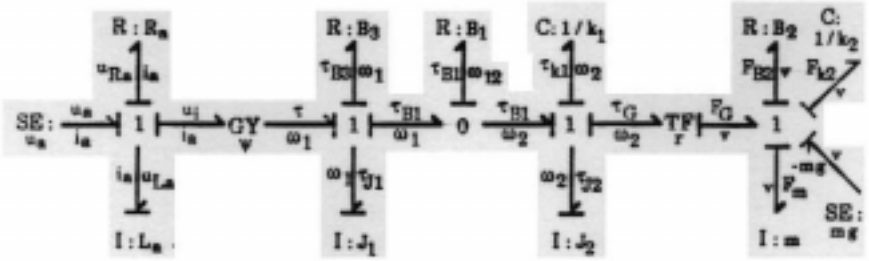

Figure 7.24. Bond graph of a DC–motor controlled mechanical system

Let us look how the causalities were assigned. We start from the left with the mandatory causality assignment for the effort source. On the 1–junction to its right, only one branch must be without a stroke. We proceed by satisfying the desired causality constraint on the "inertia" L_a, which fixes all causalities for the 1–junction. At this point in time, also the gyrator is fixed. At the next 1–junction,

we proceed in the same way, by satisfying the desired causality constraint for the inertia J_1. This fixes all causalities for that junction. At the 0–junction, we still have a choice. Since we cannot decide at once, we proceed to the next 1–junction where we satisfy the desired causality constraint for the inertia J_2. This fixes all causalities for that junction, and now also fixes the causalities for the 0–junction in between. At this point, also the following transformer is fixed. And now, we are in trouble. All causalities at the final 1–junction are already determined, and we are unable to satisfy the causality constraints for the "inertia" m. Consequently, we have detected a *structural singularity*. The system contains four "inertias" and two "compliances", and therefore, we would expect this system to be of sixth order. However, it is, in fact, only a fourth order system. We had determined, analyzed, and solved this problem already in Chapter 4 by reducing the forces that are attached to the secondary side of the transformer to its primary side. We shall not repeat the analysis at this point.

7.8 Bond Graph Modeling in DYMOLA

After we have seen how bond graphs can be constructed, let us discuss next how we can use these bond graphs to perform actual simulation runs.

The first bond graph simulation language written in the early seventies was ENPORT [7.15,7.16]. This software used an approach similar to SPICE, i.e., it did not request causalities to be specified, and it transformed the topological input description into a branch admittance matrix which could then be solved employing similar techniques to those used in SPICE. Consequently, ENPORT is able to handle structurally singular problems. The current version of the code, ENPORT–7 [7.16], offers an alphanumerical topological input language similar to SPICE, and it offers also a menu–driven graphical input language which, however, is not yet very user–friendly. A full–fledged graphical window system is currently under development. ENPORT–7 runs on various mainframe computers, but a slightly reduced version, ENPORT/PC, exists for IBM PC's and compatibles. ENPORT offers also a macro capability (somewhat comparable to the subcircuits in SPICE) which is, however, rather clumsy, and does not provide for full hierarchical decomposition capabilities.

In the late seventies, another bond graph simulation language was developed at Twente University in the Netherlands, called THT-SIM in Europe, and TUTSIM in the United States [7.17]. TUTSIM translates bond graphs into a state–space representation. The user is required to specify the causalities, and structurally singular systems cannot be handled. TUTSIM's simulation engine is somewhat poor in comparison with other state–space solvers such as ACSL. The same research group is currently prototyping a new bond graph modeling system, CAMAS [7.4], which runs on SUN's, has nice graphics capabilities, and is able to handle algebraic loops. CAMAS employs an object–oriented language (SIDOPS) for the model description which has similar properties as DYMOLA. Once available, this might become a good product.

The third product on the market is CAMP [7.8,7.9], a preprocessor to ACSL [7.13] which translates bond graphs into ACSL programs. CAMP has the same limitations as TUTSIM, i.e., it does not handle algebraic loops or structural singularities, but it has the better simulation engine (ACSL). The input format is topological (as for the two other products). It is not truly flexible with respect to handling non–standard circuit elements. Non–linear elements need to be edited manually into the generated ACSL program which is very clumsy. A graphical front end exists meanwhile also for CAMP [7.10]. However as in the case of ENPORT–7, the graphics editor is menu–driven rather than window–operated.

With the exception of the unfinished CAMAS system, none of the above products is able to handle hierarchically structured models in a general fashion which is essential for the analysis of complex systems. For these reasons, I prefer not to discuss any of these programs in greater detail, but to explain instead how DYMOLA [7.7] can be used as a bond graph modeling engine. The approach is actually straightforward. DYMOLA's "nodes" are equivalent to the 0–junctions in bond graph terminology. DYMOLA has no equivalent for 1–junctions, but as explained before, 1–junctions are the same as 0–junctions with the effort and flow variables interchanged [7.5]. Therefore, we created a model type "bond" which simply exchanges the effort and flow variables:

```
model type bond
  cut A(x/y) B(y/-x)
  main cut C[A B]
  main path P<A-B>
end
```

The bond acts just like a null–modem for a computer. Since neighboring junctions are always of opposite sex, they can both be described by regular DYMOLA "nodes" if they are connected with a "bond".

Since we don't want to maintain different types of **R**, **C**, **L**, **TF**, and **GY** elements, we add one additional rule: in DYMOLA, all elements (except for the bonds) can be attached to 0–junctions only. If they need to be attached to a 1–junction, we simply must place a bond in between.

The following DYMOLA model library suffices to describe the basic bond graphs.

```
model type SE
   cut A(e/.)
   terminal E0
   E0 = e
end

model type SF
   cut A(./ - f)
   terminal F0
   F0 = f
end

model type R
   cut A(e/f)
   parameter R = 1.0
   R * f = e
end

model type C
   cut A(e/f)
   parameter C = 1.0
   C*der(e) = f
end

model type I
   cut A(e/f)
   parameter I = 1.0
   I*der(f) = e
end
```

```
model type TF
   cut A(e1/f1) B(e2/ - f2)
   main cut C[A B]
   main path P < A - B >
   parameter m = 1.0
   e1 = m * e2
   f2 = m * f1
end

model type GY
   cut A(e1/f1) B(e2/ - f2)
   main cut C[A B]
   main path P < A - B >
   parameter r = 1.0
   e1 = r * f2
   e2 = r * f1
end
```

With these modeling elements, we can formulate a bond graph description of our simple passive circuit. Fig.7.25 shows the DYMOLA expanded bond graph with all elements being attached to 0–junctions only.

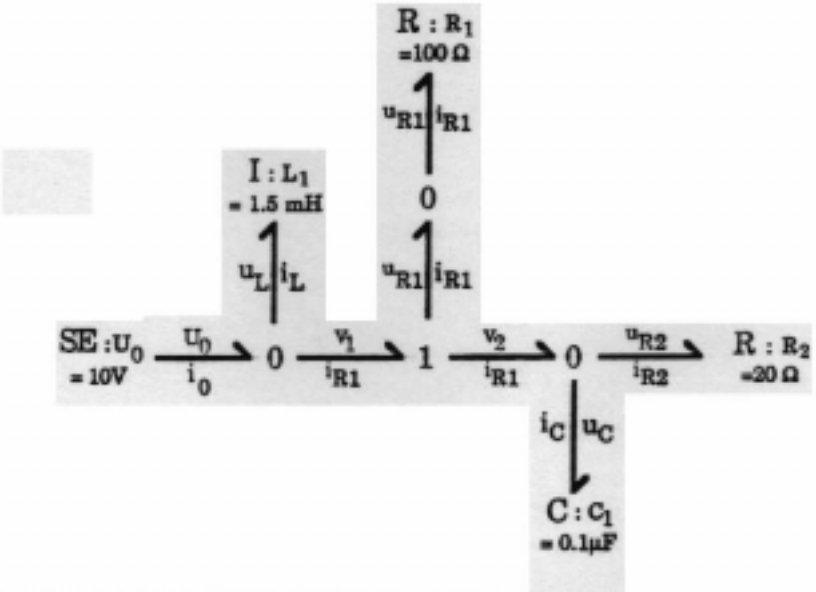

Figure 7.25. DYMOLA expanded bond graph of the passive circuit

I did not mark down the causalities here since DYMOLA is perfectly able to handle the causality assignment by itself (although no structural singularities yet).

Notice that my "bond" model type is actually a gyrator with $r = 1.0$. This special gyrator has sometimes been called *symplectic gyrator* in the bond graph literature [7.2,7.3].

This bond graph model can directly be coded in DYMOLA as shown in the code below:

```
model RLC

  submodel (SE) U0
  submodel (R) R1(R = 100.0), R2(R = 20.0)
  submodel (I) L1(I = 1.5E-3)
  submodel (C) C1(C = 0.1E-6)
  submodel (bond) B1, B2, B3
  node v1, ir1, vr1, v2
  output y1, y2

  connect U0 at      v1
  connect L1 at      v1
  connect R1 at      vr1
  connect R2 at      v2
  connect C1 at      v2
  connect B1 from    v1    to ir1
  connect B2 from    ir1   to v2
  connect B3 from    ir1   to vr1

  U0.E0 = 10.0
  y1 = C1.e
  y2 = R2.f

end
```

The interpretation of this code is straightforward.

Let us see how the DYMOLA compiler preprocesses this code. We enter DYMOLA, and specify the model to be compiled as follows:

```
$ dymola
  > enter model
    - @bond.lib
    - @rlc.dym
  > outfile rlc.eq
  > output equations
```

DYMOLA's answer is shown in the next code segment.

$U0$	$E0 = e$
$R1$	$R * f = e$
$C1$	$C * der e = f$
$L1$	$L * der f = e$
$R2$	$R * f = e$
RLC	$U0.E0 = 10.0$
	$y1 = C1.e$
	$y2 = R2.f$
	$L1.e = B1.x$
	$U0.e = L1.e$
	$C1.e = B2.y$
	$R2.e = C1.e$
	$C1.f + R2.f = B2.x$
	$B2.x = B3.x$
	$B1.y = B2.x$
	$B3.y + B2.y = B1.x$
	$R1.e = B3.y$
	$R1.f = B3.x$

We can now execute the algorithm which assigns the causalities, i.e., which determines what variable to compute from each of the equations. In DYMOLA, this is achieved with the following set of instructions:

```
> partition
> outfile rlc.sor
> output sorted equations
```

which results in the following answer:

RLC	$[U0.E0] = 10.0$
$U0$	$E0 = [e]$
RLC	$U0.e = [L1.e]$
	$L1.e = [B1.x]$
	$C1.e = [B2.y]$
	$[B3.y] + B2.y = B1.x$
	$[R1.e] = B3.y$
$R1$	$R * [f] = e$
RLC	$R1.f = [B3.x]$
	$[B2.x] = B3.x$
	$[B1.y] = B2.x$
	$[R2.e] = C1.e$
$R2$	$R * [f] = e$
RLC	$[C1.f] + R2.f = B2.x$
$C1$	$C * [der e] = f$
$L1$	$L * [der f] = e$
RLC	$[y1] = C1.e$
	$[y2] = R2.f$

The variables enclosed in "[ ]" are the variables for which each equation must be solved. This set of equations contains many trivial equations of the type $a = b$. DYMOLA is capable of throwing those out. This is accomplished through the following set of instructions:

```
> partition eliminate
> outfile rlc.sr2
> output sorted equations
```

which results in the following answer:

$R2$	$R * [y2] = y1$
RLC	$[B3.y] + y1 = 10.0$
$R1$	$R * [B3.x] = B3.y$
RLC	$[C1.f] + y2 = B3.x$
$C1$	$C * [dere] = f$
$L1$	$L * [derf] = 10.0$

which is a much reduced set of equivalent equations. The next step will be to actually perform the symbolic manipulation on the equations. In DYMOLA, this is done in the following way:

```
> outfile rlc.sov
> output solved equations
```

which results in the following answer:

$R2$	$y2 = y1/R$
RLC	$B3.y = 10.0 - y1$
$R1$	$B3.x = B3.y/R$
RLC	$C1.f = B3.x - y2$
$C1$	$dere = f/C$
$L1$	$derf = 10.0/L$

We are now ready to add the experiment description to the model. We can for instance use the one presented in Chapter 6. The set of DYMOLA instructions:

```
> enter experiment
  - @circuit.ctl
> outfile rlc.des
> output desire program
```

tells DYMOLA to generate the following DESIRE program:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-- CONTINUOUS SYSTEM RLC
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
-- STATE y1 L1$f
-- DER dC1$e dL1$f
-- PARAMETERS and CONSTANTS:
R1$R = 100.0
C = 0.1E-6
L = 1.5E-3
R2$R = 20.0
-- INITIAL VALUES OF STATES:
y1 = 0
L1$f = 0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
TMAX = 2E-5 | DT = 2E-7 | NN = 101
scale = 1
XCCC = 1
label TRY
drunr
if XCCC < 0 then XCCC = -XCCC | scale = 2 * scale | go to TRY
            else proceed
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
DYNAMIC
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - Submodel: R2
y2 = y1/R2$R
- - Submodel: RLC
B3$y = 10.0 - y1
- - Submodel: R1
B3$z = B3$y/R1$R
- - Submodel: RLC
C1$f = B3$z - y2
- - Submodel: C1
d/dt y1 = C1$f/C
- - Submodel: L1
d/dt L1$f = 10.0/L
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
OUT
dispt y1, y2
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
/--
/PIC 'rlc.PRC'
/--
```

which can be executed at once using the following instructions:

```
  > stop
$ desire 0
  > load 'rlc.des'
  > run
```

which will immediately (within less than a second) produce the desired output variables u_C, and i_{R2} on the screen. Both DYMOLA [7.7] and DESIRE [7.12] are currently running alternatively on VAX/VMS or PC/MS–DOS. The code will run fine on a PC/XT, but minimum requirements are a 10 *MByte* hard disk and an 8087 co–processor. Faster versions exist for the PC/AT, and the 386–based machines. DESIRE [7.12] supports CGA, EGA, and VGA graphics.

7.9 The Dual Bond Graph

In some cases, bond graphs may result which have many more elements attached to 1–junctions than to 0–junctions. By using the previously introduced methodology, this would force us to introduce many additional 0–junctions and bonds in order to be able to attach all elements to 0–junctions only.

It is possible to circumvent this problem by introducing the concept of the dual bond graph [7.5]. For any bond graph, an equivalent dual bond graph exists in which the role of all effort and flow variables is interchanged. Table 7.2 illustrates what happens to the various bond graph elements under the transition from the regular to the dual bond graph.

Table 7.2 Relation between regular and dual bond graph

regular bond graph	dual bond graph
e	f
R	G
C	L
SE	SF
TF	TF
GY	GY
0–junction	1–junction

In Fig.7.26, this duality transformation has been applied to the bond graph of Fig.7.24.

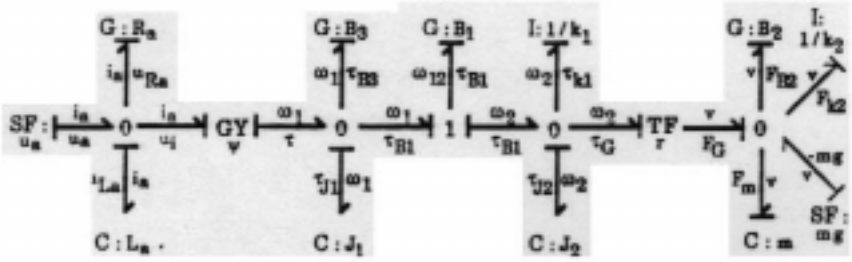

Figure 7.26. Dual bond graph of the electro–mechanical system

It can be easily verified that the equations generated from the dual bond graph are exactly the same as those that were generated from the regular bond graph. However, in the regular bond graph, all elements except for the friction element B_1 were attached to 1–junctions. This would have forced us to create 10 additional junctions with 10 additional bonds (one for each element attached to a 1–junction). In the dual bond graph, all elements except for the friction B_1 are attached to 0–junctions. Consequently, we need to expand the bond graph only with one additional 0–junction and bond instead of the former 10.

We just introduced a "new" bond graph element: the *conductance* G. Its governing equation is: $f = G \cdot e$, which DYMOLA can, of course, transform into $e = \frac{1}{G} \cdot f$. This element is non–essential. Instead of replacing the resistors with conductances in the dual bond graph and writing for instance "G:R_a" in Fig.7.26, we could equally well have kept the resistors and could have written "R:$\frac{1}{R_a}$" in Fig.7.26. It just seemed more convenient this way. Notice that this decision has absolutely nothing to do with the assumed causality. Both *resistors* and *conductances* can assume either causality.

Notice that the duality transformation can also be applied to subsystems only. Natural places where the bond graph can be cut into subsystems are the transformers and gyrators where one sort of energy is transformed into another. Fig.7.27 shows yet another version of the same system. This time, only the electrical subsystem has been transformed to its dual equivalent.

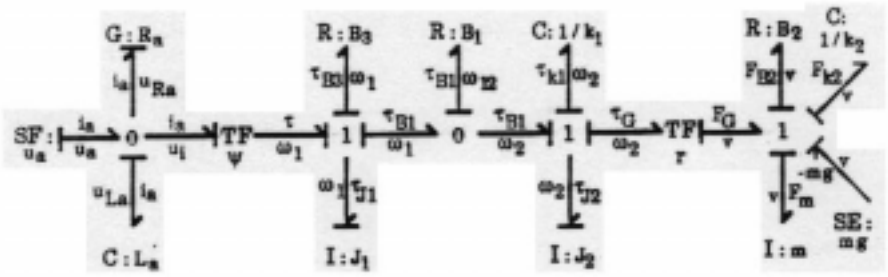

Figure 7.27. Partially transformed bond graph of the mechanical system

In this duality transformation, gyrators become transformers and vice versa. The reason why the bond graph of the DC–motor exhibits a *gyrator* at the interface between its electrical and its mechanical subsystem is because of the peculiar way in which Paynter defined his "efforts" and his "flows" for mechanical and for electrical systems [7.14].

However, it is possible to cut systems also at other places. Let me illustrate the concept by means of a series of snapshots of a portion of the previously used bond graph of Fig.7.26. Fig.7.28a shows the portion that we want to concentrate on.

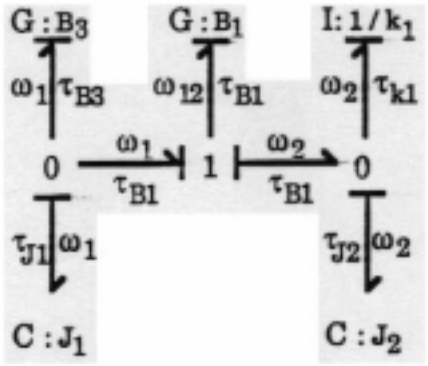

Figure 7.28a. Portion of the DC–motor bond graph

In Fig.7.28b, the two 0–junctions have been stretched out into two junctions each. The arrowless line between the two 0–junctions indicates that this is, in fact, the same junction. The solid line symbolizes a "wire" that exists between the two junctions.

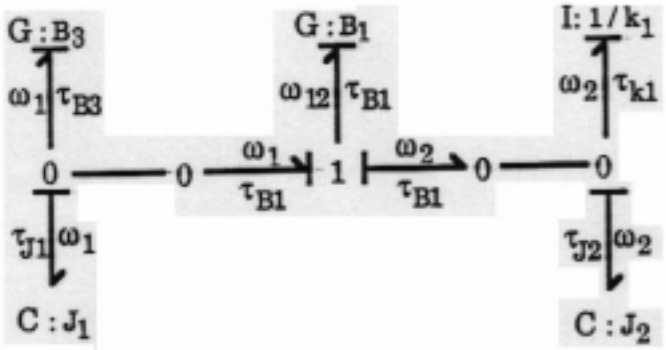

Figure 7.28b. Expanded bond graph

Obviously, any such "wire" can be replaced by a "symplectic transformer", i.e., a transformer with $m = 1$. Such a transformer is obviously equivalent to a "wire". This replacement is shown in Fig.7.28c.

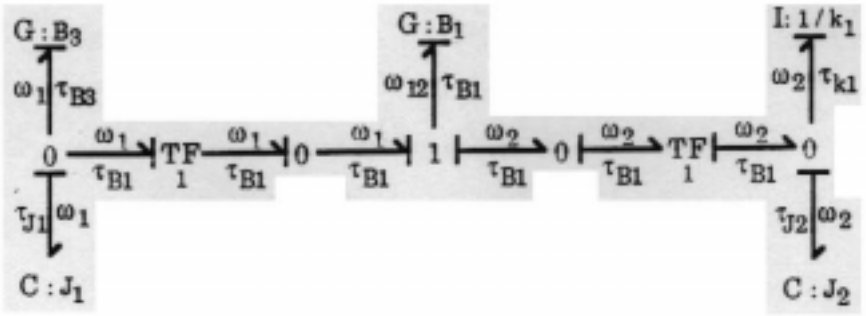

Figure 7.28c. Further expanded bond graph

At this point, we see that the bond graph contains two 0–junctions with two connections only. These junctions are unimportant and can be eliminated. This is shown in Fig. 7.28d.

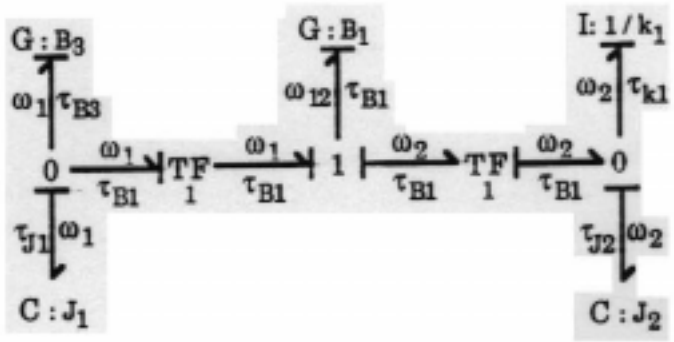

Figure 7.28d. Reduced bond graph

Now, we have two transformers in the circuit which isolate the portion of the circuit between them. We can, thus, apply a duality transformation to that portion of the circuit. This is shown in Fig.7.28e.

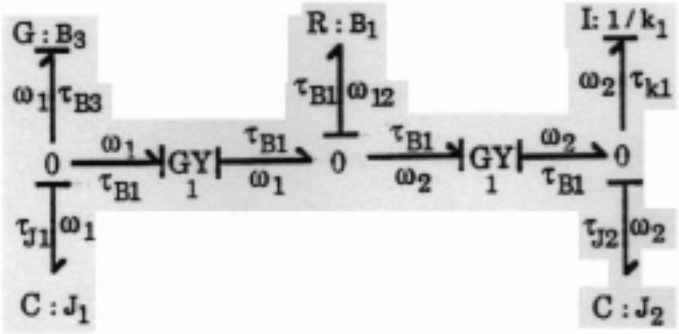

Figure 7.28e. Dually transformed bond graph

The two symplectic transformers have been transformed into two symplectic gyrators which, of course, are the same as our Dymola bonds. I still prefer to leave these gyrators explicitly in the circuit and use the *GY.dym* model rather than the bond.dym model since a reduction to a normal bond might be graphically confusing. However, we just learned that a bond graph can be cut into subsystems at an arbitrary bond. In the duality transformation, the cutting bond (an implicit symplectic gyrator) is transformed into an explicitly shown symplectic gyrator.

7.10 Summary

In this chapter, we have looked at a number of different graphical modeling techniques, and we have analyzed one among them, the bond graph modeling technique, in greater detail. It was the aim of this chapter to relate this seemingly quite different approach to modeling back to the previously introduced methodologies and terminologies.

Future chapters will make more references to bond graphs, and, in particular, Chapter 8 will discuss the application of bond graphs to non–equilibrium state thermodynamics. This application will provide us with even more motivation for the bond graph methodology as a whole.

References

[7.1] Alan Blundell (1982), *Bond Graphs for Modelling Engineering Systems*, Ellis Horwood Publishers, Chichester, United Kingdom, and Halsted Press, New York.

[7.2] Pieter C. Breedveld (1982), "Thermodynamic Bond Graphs and the Problem of Thermal Inertance", *J. Franklin Institute*, **314**(1), pp. 15–40.

[7.3] Pieter C. Breedveld (1984), *Physical Systems Theory in Terms of Bond Graphs*, Ph.D. Dissertation, Technical University Twente, Enschede, The Netherlands.

[7.4] Jan F. Broenink (1990), *Computer–Aided Physical–Systems Modeling and Simulation: A Bond–Graph Approach*, Ph.D. Dissertation, Universiteit Twente, Enschede, The Netherlands.

[7.5] François E. Cellier (1990), "Hierarchical Non-linear Bond Graphs — A Unified Methodology for Modeling Complex Physical Systems", Keynote Address, *Proceedings European Simulation MultiConference*, Nürnberg, FRG, pp. 1–13.

[7.6] Hilding Elmqvist (1975), *SIMNON – An Interactive Simulation Program for Non–linear Systems — User's Manual*, Report CODEN: LUTFD2/(TFRT–7502), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[7.7] Hilding Elmqvist (1978), *A Structured Model Language for Large Continuous Systems*, Ph.D. Thesis, Report CODEN: LUTFD2/(TRFT–1015), Dept. of Automatic Control, Lund Institute of Technology, Lund, Sweden.

[7.8] Jose J. Granda (1982), *Computer Aided Modeling Program (CAMP): A Bond Graph Preprocessor for Computer Aided Design and Simulation of Physical Systems Using Digital Simulation Languages*, Ph.D. Dissertation, Dept. of Mechanical Engineering, University of California, Davis.

[7.9] Jose J. Granda (1985), "Computer Generation of Physical System Differential Equations Using Bond Graphs", *J. Franklin Institute*, **319**(1/2), pp. 243–255.

[7.10] Jose J. Granda, and F. Pourrahimi (1985), "Computer Graphic Techniques for the Generation and Analysis of Physical System Models", in: *Artificial Intelligence, Graphics, and Simulation*, Proceedings of the Western Simulation MultiConference, (G. Birtwistle, ed.), SCS Publishing, pp. 70–75.

[7.11] Dean C. Karnopp, and Ronald C. Rosenberg (1974), *System Dynamics; A Unified Approach*, John Wiley, New York.

[7.12] Granino A. Korn (1989), *Interactive Dynamic-System Simulation*, McGraw–Hill, New York.

[7.13] Edward E.L. Mitchell, and Joseph S. Gauthier (1986), *ACSL: Advanced Continuous Simulation Language — User Guide / Reference Manual*, Mitchell & Gauthier Assoc., 73 Junction Square, Concord, MA 01742.

[7.14] Henry M. Paynter (1961), *Analysis and Design of Engineering Systems*, M.I.T. Press, Cambridge, MA.

[7.15] Ronald C. Rosenberg (1974), *A User's Guide to ENPORT-4*, John Wiley, New York.

[7.16] RosenCode Associates, Inc. (1989), *The ENPORT Reference Manual*, RosenCode Associates, Inc., 200 N. Capitol Bldg., Lansing, MI 48933.

[7.17] Jan J. van Dixhoorn (1982), "Bond Graphs and the Challenge of a Unified Modelling Theory of Physical Systems", in: *Progress in Modelling and Simulation*, (F. E. Cellier, ed.), Academic Press, London, pp. 207–245.

Bibliography

[B7.1] Albert M. Bos, and Pieter C. Breedveld (1985), "Update of the Bond Graph Bibliography", *J. Franklin Institute*, **319**(1/2), pp. 269–286.

[B7.2] Vernon D. Gebben (1979), "Bond Graph Bibliography", *J. Franklin Institute*, **308**(3), pp. 361–369.

[B7.3] Louis P. A. Robichaud (1962), *Signal Flow Graphs and Applications*, Prentice–Hall, Englewood Cliffs, N.J.

Homework Problems

[H7.1] Algebraic Loop

Draw a bond graph for the simple resistive circuit of Fig.6.11. To refresh your memory, the circuit is shown once more in Fig.H7.1.

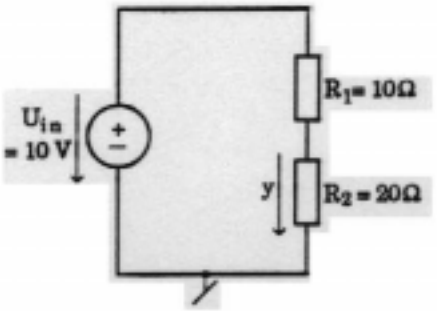

Figure H7.1. Schematic of a trivial resistive circuit

Prove that this representation contains an algebraic loop by showing that you have a choice in assigning the causalities. Create a DYMOLA program that implements this bond graph, and show that, during the partitioning process, the set of equations which initially looks quite different from the one obtained in Chapter 6, is reduced to the same set of three algebraically coupled equations that we came across in Chapter 6.

[H7.2] Electrical Circuit

Solve hw(H6.6) once more this time using the bond graph approach. The circuit is presented again in Fig.H7.2.

Since, also in the bond graph library, the sources have been declared as terminals, the dependent current source can be treated in the DYMOLA program exactly in the same manner as in Chapter 6. Since the cause/effect relationship between the driving voltage and the driven current contains only one rather than two variables, this connection is not a bond. It is a regular signal path (as in a block diagram), and, also in the bond graph, it is represented through a *thin full arrow* that emanates from the 0–junction at which all bonds have the driving voltage as their effort variable, and that ends at the SF element that is being driven by this signal.

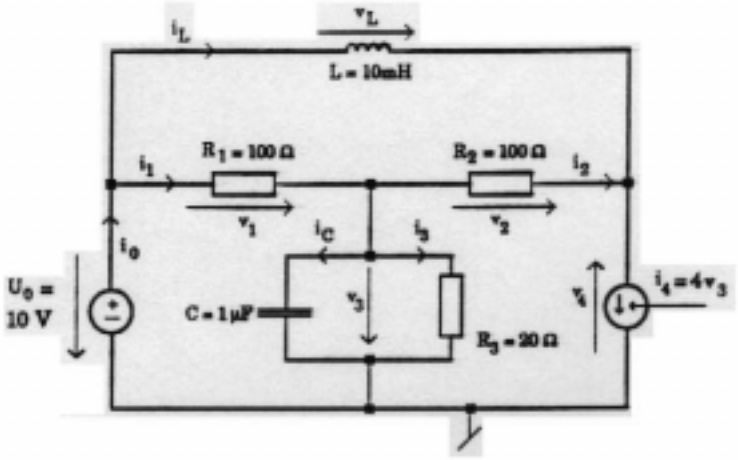

Figure H7.2. Circuit diagram of a simple passive circuit

[H7.3] Mechanical System

Solve hw(H4.3) once more this time using bond graphs. Fig.H7.3 shows the system again.

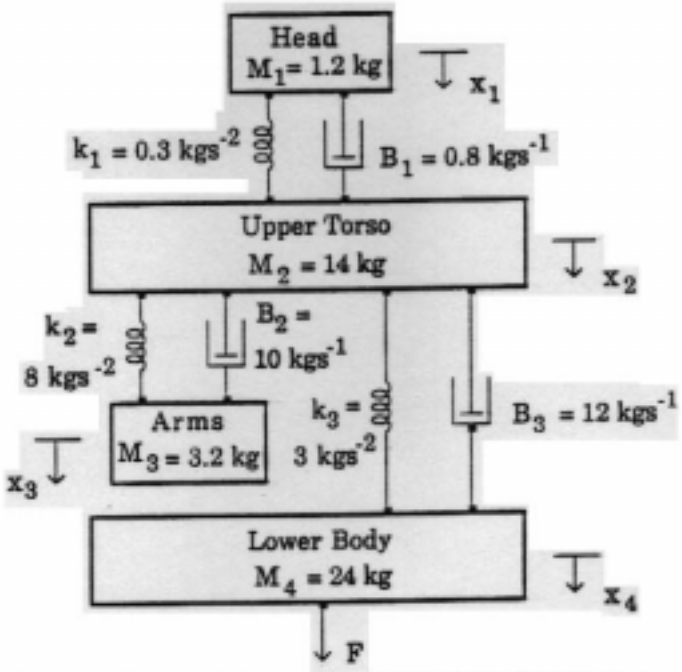

Figure H7.3. Mechanical model of a sitting human body

Create a bond graph for this system, transform the bond graph into a DYMOLA program, and generate either a SIMNON [7.6] or a DESIRE [7.12] program from it. Contrary to the simulation of Chapter 4, this time, we shall simulate the system in the time domain for a sinusoidal input of varying frequency. In general, a sinusoidal input of any frequency ω_0 can be written as:

$$u = \sin(\omega_0 t) \qquad (H7.3a)$$

In our experiment, we wish to vary ω_0 using a slow ramp of time, i.e.:

$$\omega_0 = k \cdot t \qquad (H7.3b)$$

By plugging eq(H7.3b) into eq(H7.3a), we find:

$$u = \sin(k \cdot t^2) \qquad (H7.3c)$$

Set $k = 0.01$, and simulate the system during 100 *sec*. Use a step size of 0.01 *sec*. Observe the input and the output over time.

[H7.4] Electro–Mechanical System

For the system shown in Fig.H7.4, generate a bond graph, assign the proper causalities, and discuss the energy flow through the system.

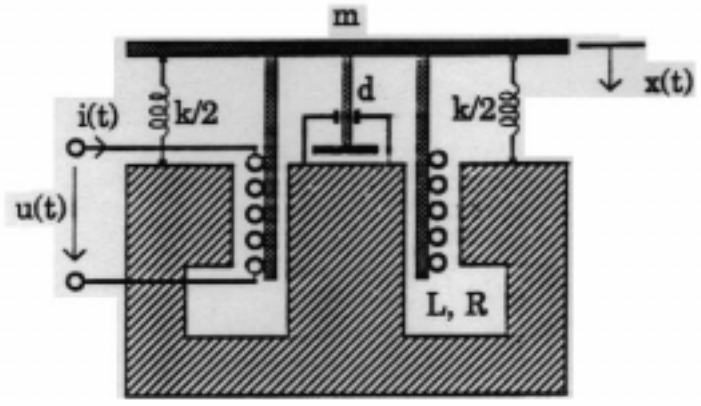

Figure H7.4. Electro–mechanical system

This system had once before been discussed as hw(H4.4). Use a duality transformation prior to coding the bond graph in DYMOLA, and generate a minimal state–space model using the DYMOLA preprocessor. Compare the resulting model with the one that you had found in hw(H4.4).

[H7.5] Non–Ideal Transformer

Fig.H7.5a shows a non–ideal transformer, and Fig.H7.5b shows an equivalent circuit that reduces the transformer to elements which we already know.

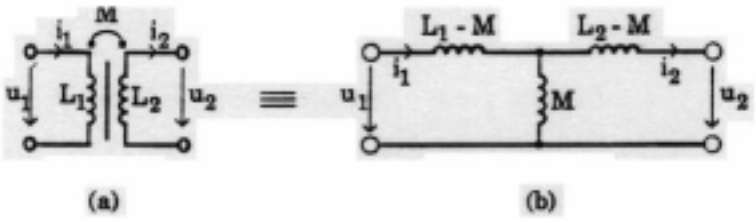

Figure H7.5. Non–ideal transformer

Create a bond graph for the transformer using the equivalent circuit approach. Introduce the causalities, and determine that the system contains a structural singularity. Extract the equations from the bond graph, and verify the structural singularity by means of these equations. Manually reduce the structural singularity. This will unfortunately introduce an algebraic loop. Reduce the algebraic loop, and determine a state–space model that describes the non–ideal transformer. Create a DYMOLA model (not a bond graph model) which describes this executable set of equations, but assign cuts to this model such that the model can be used as a component model type (named RTF) anywhere within a bond graph model.

[H7.6] Hydraulic System

Fig.H7.6a shows a schematic diagram of a hydraulic motor with a four–way servo valve. The input to this system is the position of the piston, x. If the piston of the servo valve is moved far to the right, then the pressure p_1 in the first chamber is the same as the high pressure P_S, and the pressure p_2 of the second chamber is the same as the low pressure P_0. Consequently, the hydraulic motor will wish to increase the volume of the first chamber and decrease the volume of the second by moving the motor block to the right. In the given setup, the axis of the hydraulic motor is a screw which therefore starts rotating. If the piston of the servo valve is moved far to the left, then the pressure p_1 is equal to the low pressure P_0, the pressure p_2 is equal to the high pressure P_S, and the hydraulic motor rotates in the opposite direction. In between, one position exists where $p_1 = p_2$. In that position, the hydraulic motor does not move. We call this position $x = 0.0$. Hydraulic motors are able to move large masses with little control power, and yet, the inertia of the motor is much smaller than for an equivalent electrical motor.

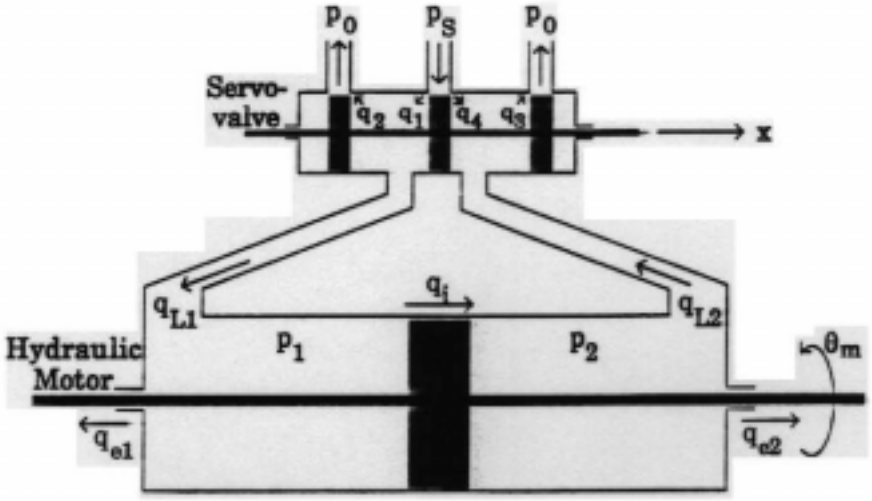

Figure H7.6a. Hydraulic motor with a four-way servo valve

The flows from the high pressure line into the servo valve and from the servo valve back into the low pressure line are turbulent. Consequently, the relation between flow and pressure is quadratic:

$$q_1 = k(x_0 + x)\sqrt{P_S - p_1} \qquad (H7.6a)$$

$$q_2 = k(x_0 - x)\sqrt{p_1 - P_0} \qquad (H7.6b)$$

$$q_3 = k(x_0 + x)\sqrt{p_2 - P_0} \qquad (H7.6c)$$

$$q_4 = k(x_0 - x)\sqrt{P_S - p_2} \qquad (H7.6d)$$

For our servo valve, we want to use the following parameter values: $P_S = 0.137 \times 10^8\ N\ m^{-2}$, $P_0 = 0.0\ N\ m^{-2}$, $x_0 = 0.05\ m$, and $k = 0.248 \times 10^{-6}\ kg^{-\frac{1}{2}}\ m^{\frac{5}{2}}$. You have to be a little careful with these equations since the expressions $x_0 \pm x$ must never become negative, and the same is true for the expressions under the square roots. A negative pressure difference is even physically possible due to *cavitation*, but our model does not represent cavitation phenomena properly. In your computer programs, you better limit these expressions explicitly.

The flow q_i is an internal leakage flow which is laminar, and therefore linear in the load pressure p_L, i.e.

$$q_i = c_i \cdot p_L = c_i(p_1 - p_2) \qquad (H7.6e)$$

where $c_i = 0.737 \times 10^{-13}\ kg^{-1}\ m^4\ sec$. The flows q_{e1} and q_{e2} are external leakage flows which are also laminar:

$$q_{e1} = c_e \cdot p_1 \qquad (H7.6f)$$
$$q_{e2} = c_e \cdot p_2 \qquad (H7.6g)$$

where $c_e = 0.737 \times 10^{-12}\ kg^{-1}\ m^4\ sec$.

The change in the chamber pressures is proportional to the effective flows in the two chambers:

$$\dot{p}_1 = c_1(q_{L1} - q_i - q_{e1} - q_{ind}) \qquad (H7.6h)$$
$$\dot{p}_2 = c_1(q_{ind} + q_i - q_{e2} - q_{L2}) \qquad (H7.6i)$$

where $c_1 = 5.857 \times 10^{13}\ kg\ m^{-4}\ sec^{-2}$. q_{ind} is the *induced flow* which is a result of the moving motor block:

$$q_{ind} = \psi \cdot \dot{\theta}_m \qquad (H7.6j)$$

and it is proportional to the angular velocity of the hydraulic motor. In our case $\psi = 0.575 \times 10^{-5}\ m^3$.

The torque produced on the hydraulic motor is proportional to the load pressure p_L:

$$T_m = \psi \cdot p_L = \psi(p_1 - p_2) \qquad (H7.6k)$$

On the mechanical side of the motor, we have an inertia of $J_m = 0.08\ kg\ m^2$, and a viscous friction of $\rho = 1.5\ kg\ m^2\ sec^{-1}$.

Fig.H7.6b shows the overall position control circuit.

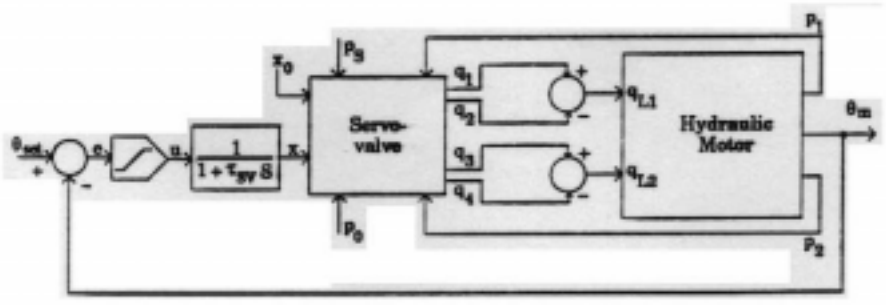

Figure H7.6b. Hydraulic motor position control circuit

The first order controller represents the translation from the (electrical) control signal u to the (mechanical) position of the piston of the servo valve. $\tau_{sv} = 0.005$ *sec* is the mechanical time constant of the servo valve. The error signal e is limited between -1.0 and +1.0.

Generate a block diagram that describes the overall system. Code this position control problem in any CSSL, and simulate the step response of this system during 0.2 *sec*.

[H7.7] Hydraulic System

For the system of hw(H7.6), generate a bond graph description. The control signals are represented by signal paths, but the hydraulic and the mechanical parts can be easily described in terms of a bond graph. Discuss the energy flow in this system. Code the bond graph model in DYMOLA, generate either a DESIRE [7.12] or a SIMNON [7.6] program, and simulate the step response of this system during 0.2 *sec*. Compare the results with those found in hw(H7.6).

[H7.8]* Surge Tank Simulation

In hw(H5.4), we have already once analyzed the behavior of a water turbine with a surge tank. To refresh your memory, the schematic of that system is shown again in Fig.H7.8.

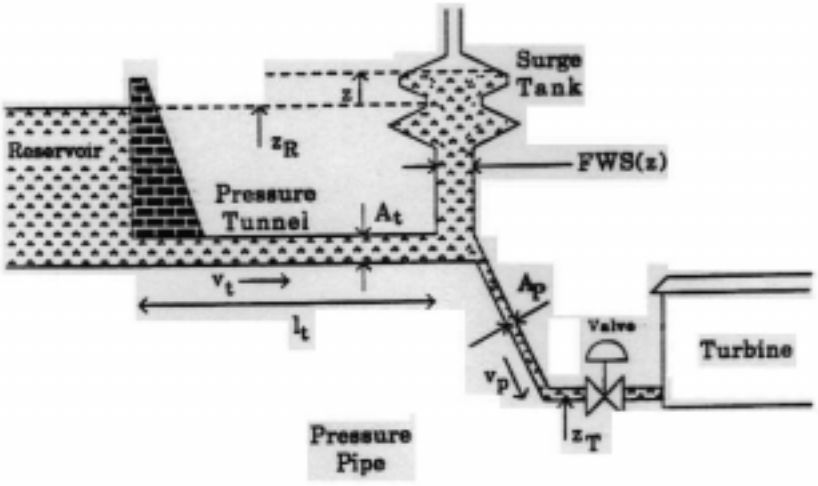

Figure H7.8. Power generation using a water turbine

This time, we wish to represent this system using a bond graph notation. This will ensure that we have modeled the energy flow through the system

correctly. For this purpose, we need to know one more piece of information which was not required for the previous model: the pressure tunnel is at an altitude of 3100 m.

Code your bond graph model in DYMOLA, generate either a DESIRE [7.12] or a SIMNON [7.6] program, and simulate the system. Compare your results with those found in hw(H5.4).

Projects

[P7.1] Code Generation

Develop a new code generator for DYMOLA for the generation of ACSL [7.13] programs in addition to the currently implemented DESIRE [7.12] and SIMNON [7.6] interfaces.

Research

[R7.1] Graphical Input

Develop a graphical front end for bond graph models. Using this software, we should be able to draw bond graphs on the screen using a zoom and pan capability. We should also be able to assign new bond graph symbols (icons) to subnetworks for hierarchical modeling of bond graphs. Finally, we should be able to describe new atomic bond graph models as DYMOLA model types and assign these model types to new bond graph icons. A graphical preprocessor is to be developed which can translate the schematic into a DYMOLA program.