Model Based Development of Embedded Systems 2014

Modeling in Simulink
Outline

- Modeling dynamic system
- Simulink basics
  - Block
  - Subsystem
  - Sample time
  - Solver
  - Algebraic loops
- Simple examples
Modeling Dynamic System

- Most of the real-life systems are dynamic systems.
- Dynamic system is an entity or object that is "excited" by external input stimuli ("inputs") and produces a response ("outputs").
- Input and outputs are signals of time.
- Continuous, Discrete and Hybrid.

```
inputs  
\[ u(t) \]  
```
```
System  
```
```
outputs 
\[ y(t) \]  
```
Modeling Dynamic System

- A mathematical model of a dynamic system is described by a set of equations.
- At any given instant of time, these equations may be viewed as relationships between the system's output, the system's input stimuli at that time, the current state of the system, the system parameters, and time.
- The state of the system may be thought of as a numerical representation of the dynamically changing configuration of the system.
Modeling Dynamic System

- Differential equations
- Difference equations
- Algebraic equations
- Composite

Ordinary differential equations (ODEs):

\[
\begin{align*}
\text{inputs} & : u_c(t) \\
\text{System States: } & x_c(t), \text{ Parameters: } P \\
\text{Output: } & y_c(t) = f_c(t, x_c(t), u_c(t), P) \\
\text{Derivative: } & \dot{x}_c(t) = g(t, x_c(t), u_c(t), P) \\
\text{outputs} & : y_c(t)
\end{align*}
\]

Difference equations:

\[
\begin{align*}
\text{inputs} & : u_d(t_k) \\
\text{System States: } & x_d(t_k), \text{ Parameters: } P \\
\text{Output: } & y_d(t_k) = f_d(t, x_d(t_k), u_d(t_k), P) \\
\text{Update: } & x_d(t_{k+1}) = h(t, x_d(t_k), u_d(t_k), P) \\
\text{outputs} & : y_d(t_k)
\end{align*}
\]

Algebraic equations:

\[
\begin{align*}
\text{inputs} & : u_a(t) \\
\text{System Parameters: } & P \\
\text{Output: } & y_a(t) = f_a(t, u_a(t), u_a(t), P) = 0 \\
\text{outputs} & : y_a(t)
\end{align*}
\]

Composite system:

\[
\begin{align*}
\text{inputs} & : u_c(t), u_d(t), u_a(t) \\
\text{System States: } & x(t), \text{ Parameters: } P \\
\text{Output: } & y(t) = f(t, x(t), u(t), P), f_a(t, u(t), y_a(t), x(t), P) = 0 \\
\text{Update: } & x_d(t_{k+1}) = h(t, x(t), u(t), P) \\
\text{Derivative: } & \dot{x}_c(t) = g(t, x(t), u(t), P) \\
\text{outputs} & : y_c(t), y_d(t_k), y_a(t)
\end{align*}
\]

\[
x(t) = \begin{bmatrix} x_c(t) \\ x_d(t_k) \end{bmatrix}, \quad u(t) = \begin{bmatrix} u_c(t) \\ u_d(t_k) \\ u_a(t) \end{bmatrix}, \quad y(t) = \begin{bmatrix} y_c(t) \\ y_d(t_k) \\ y_a(t) \end{bmatrix}
\]
Dynamic System - example

- Modeling a single degree of freedom system using 2nd order differential equation

\[ \ddot{x} = \frac{1}{m} [f(t) - c\dot{x} - kx] \]

The SDOF Mass-Spring-Dashpot.
Simulink

- Simulink is a Modeling language
- Uses **Synchronous Reactive Modeling**
- A Simulink block diagram model is a graphical representation of a mathematical model of a dynamic system

\[ S = \{b_1, b_2, \ldots, b_n\} \]

- Purely functional: No real timing
  - Computations performed in “zero” simulated time
  - Pragmatically: the system response is guaranteed to be completed before the next system event
  - In a block, outputs and state are computed immediately when a block is triggered
Simulink Block

- Blocks can be regular (dataflow) or Stateflow (state machine)
- Regular blocks can be discrete or continuous
- All blocks work on continuous signals, discrete blocks are activated by events
- Blocks can be stateless or with state
Dynamic System - example

- Let's see example of the spring damper

\[ \ddot{x} = \frac{1}{m} \left[ f(t) - c\dot{x} - kx \right] \]
Simulation Loop

1. Initialize
2. Time < StopTime?
   - no: Simulation complete
   - yes: ModelOutputs → Algebraic Loop Solver → ModelUpdate
3. Numerical Integration: \( X_c = \int \dot{X}_c \)
4. ModelOutputs
5. Algebraic Loop Solver
6. ModelDerivatives, \( \dot{X}_c \)
7. Zero Crossing Detector
8. Time = Time + StepSize

Single Tasking (Including Variable Step) Simulation Loop
Direct Feedthrough and Sort order

- Computing the output of a block may need the data at input.
- In this case, the input port is of type "direct feedthrough"
- Examples of direct feedthrough blocks
  - Elementary math block, gain, product, etc
- In Model Compilation, necessary to define order of operations
- Simulink sorts the blocks so that blocks w. direct feedthrough inputs are evaluated after their source blocks. Other blocks have not specified order
- Order can be user-specified by putting Block Priorities
- The lower the number, the higher the priority (Negative numbers and 0 are valid priority values)
Block Update

Feedthrough

Not Feedthrough
Simulation Time

- Selecting properly the total simulation time is important for the physical insight obtained for a simulated dynamic model.
- Improper time selection may lead to erroneous or incomplete conclusions.
- Time constants of dynamic subsystems should be considered for selecting a simulation time frame that will capture the transient and the steady state regimes.
- For example, a short simulation time can only reveal transient behaviors and not return the steady state condition of a stable dynamic system.
- The minimum total simulation time required is usually determined by the subsystem/component that has the slowest dynamics (largest time constant).
Algebraic Loops

- Algebraic Loop = output goes directly to direct feedthrough input
- Algebraic loops arise when a model includes an algebraic constraint, \( F(z) = 0 \)

Simulink tries to solve algebraic loops, but can lead to undefined behavior

Let us do an example

Non Algebraic direct feedthrough loops
- loops involving triggered subsytem
- A loop from the output to the reset port of an integrator
Algebraic Loops

- Solution: use Delay, Integrator or other history related block
- Algebraic Constraint blocks constrains its input signal $F(z)$ to zero and outputs an algebraic state $z$

\[
\begin{align*}
&z_2 + z_1 = 1 \\
&z_2 - z_1 = 1
\end{align*}
\]
**Subsystem**

- Reduce blocks displayed in a model window
- Keep functionally related block together
- Establish hierarchical block diagram

*Subsystems can be*

  *Virtual*
  *Atomic or non-virtual*
  *Enabled, triggered or both*
Atomic Subsystem

Represent nonvirtual systems within another system

Have their own sampling rate

Have their own code generating characteristics

Have their own execution order number
Masking Subsystem

Mask - Encapsulation with a UI

Provides

- Mask icon display
- Block description
- Parameter dialog prompt
- Custom block help text
Enabled and Triggered Subsystem

- Enabled subsystem is active when a special control signal is active
- Triggered subsystem is executed only when a control signal changes
  note: Sample time can be only constant or inherited.
Solver Selection

- Mathematicians have developed a wide variety of numerical integration techniques for solving the ordinary differential equations (ODEs) that represent the continuous states of dynamic systems.
- Simulink simulates a dynamic system by computing its states at successive time steps over a specified time span.
- Simulink provides an extensive set of fixed-step and variable-step continuous solvers, each implementing a specific ODE solution method.
Solver Selection

- Determines solution at current time step
- Determines the next simulation time step

Solver options:

**Fixed-Step**
- Ode1
- Ode2
- Ode3
- Ode4
- Ode5
- Ode8

**Variable-Step**
- Ode45
- Ode23
- Ode113
- Ode15s
- Ode23s
- Ode23t
- Ode23tb
Solver Selection

- **Fixed-step solvers**
  - Solve the model at regular time intervals (time step size) from the beginning to the end of the simulation
  - Select a step size and hence simulation rate fast enough to track state changes in the fastest block in your model
  - Generally, decreasing the step size increases the accuracy of the results while increasing the time required to simulate the system

- **Variable-step solvers**
  - Vary the step size during the simulation
  - Reduce the step size to increase accuracy when a model's states are changing rapidly
  - Increase the step size to avoid taking unnecessary steps when the model's states are changing slowly
Solver Selection

- **Continuous solvers**
  
  Use numerical integration to compute a model's continuous states at the current time step from the states at previous time steps and the state derivatives.
  
  Rely on the model's blocks to compute the values of the model's discrete states at each time step.

- **Discrete solvers**
  
  Discrete solvers exist primarily to solve purely discrete models.
  
  They compute the next simulation time step for a model.
  
  They do not compute continuous states.
  
  They rely on the model's blocks to update the model's discrete states.
Variable Step size

\[ mx\_step\_size = \frac{t_{\text{stop}} - t_{\text{start}}}{\text{time \_ steps}} \]

Max step size is the largest time step that the solver can take (‘auto’ means 50 steps)
Sample Time

- **Sample time** specifies how often and when the block needs to be evaluated for computing its output.
- A **continuous** block has an infinitesimal sample time.
- Many blocks allow to specify "sample time" in Mask:
  - $T_s$ sample time
    - $t_n = n*T_s$
    - $T_s = 0$ means "continuous"
    - $T_s = \infty$ means constant Sample Time
    - $T_s = -1$ means "inherited"
  - $[T_s,T_o]$ sample time with offset
    - $t_n = n*T_s + T_o$
- Sample time of blocks determines the **fundamental sample time**
  - GCD of sample times in system.
- In **fixed-step** simulation, auto step size gives fundamental sample time.
- In **variable-step** simulation, step is adjusted to find **sample hit**.
Under “Discontinuous” you will find coulomb friction, dead zone, saturation, and relay.

Under “Continuous” there is a delay block.

Under “Simulink Extras” there is a PID controller, transfer function with non-zero initial conditions, some useful sinks (such as power spectral density), and radians-to-degrees and Fahrenheit-to-Celsius converters.
Another Example

- Cruise control

![Cruise control diagram](http://ctms.engin.umich.edu/CTMS/index.php?example=CruiseControl&section=SimulinkModeling)
Explore

- [http://www.eng.ox.ac.uk/~labejp/Seminar/Simulink/Exercises.pdf](http://www.eng.ox.ac.uk/~labejp/Seminar/Simulink/Exercises.pdf)
- [http://128.173.204.63/courses/matlab/matlab_tutorial.html](http://128.173.204.63/courses/matlab/matlab_tutorial.html)
- Embedded systems in car!

[Car diagram with various systems labeled]

- [http://www.cvel.clemson.edu/auto/systems/auto-systems.html](http://www.cvel.clemson.edu/auto/systems/auto-systems.html)
Next

- Modeling discrete systems
- Multirate system
- Library and custom blocks
- Function calls
- S function
- Embedded code generation
- More complex examples