**The lecture 8**

**Modelling control flow blocks**

**Creating Conditionally Executed Subsystems**

A *conditionally executed subsystem* is a subsystem whose execution depends on the value of an input signal. The signal that controls whether a subsystem executes is called the *control signal*. The signal enters the Subsystem block at the *control input*.

Conditionally executed subsystems can be very useful when you are building complex models that contain components whose execution depends on other components.

Simulink supports the following types of conditionally executed subsystems:

**•**An *enabled subsystem* executes while the control signal is positive. It starts execution at the time step where the control signal crosses zero (from the negative to the positive direction) and continues execution while the control signal remains positive.

**•**A *triggered subsystem* executes once each time a trigger event occurs. A trigger event can occur on the rising or falling edge of a trigger signal, which can be continuous or discrete.

**•**A *triggered and enabled subsystem* executes once on the time step when a trigger event occurs if the enable control signal has a positive value at that step.

**•**A *control flow subsystem* executes one or more times at the current time step when enabled by a control flow block that implements control logic similar to that expressed by programming language control flow statements.

**Enabled Subsystems**

Enabled subsystems are subsystems that execute at each simulation step where the control signal has a positive value.

An enabled subsystem has a single control input, which can be scalar or vector valued.

**•**If the input is a scalar, the subsystem executes if the input value is greater than zero.

**•**If the input is a vector, the subsystem executes if *any* of the vector elements is greater than zero.

For example, if the control input signal is a sine wave, the subsystem is alternately enabled and disabled, as shown in this figure. An up arrow signifies enable, a down arrow disable.



Simulink uses the zero-crossing slope method to determine whether an enable is to occur. If the signal crosses zero and the slope is positive, the subsystem is enabled. If the slope is negative at the zero crossing, the subsystem is disabled.

**Creating an Enabled Subsystem**

You create an enabled subsystem by copying an Enable block from the Signals & Systems library into a subsystem. Simulink adds an enable symbol and an enable control input port to the Subsystem block.



**Setting Output Values While the Subsystem Is Disabled.** Although an enabled subsystem does not execute while it is disabled, the output signal is still available to other blocks. While an enabled subsystem is disabled, you can choose to hold the subsystem outputs at their previous values or reset them to their initial conditions.

**Blocks an Enabled Subsystem Can Contain**

An enabled subsystem can contain any block, whether continuous or discrete. Discrete blocks in an enabled subsystem execute only when the subsystem executes, and only when their sample times are synchronized with the simulation sample time. Enabled subsystems and the model use a common clock.

For example, this system contains four discrete blocks and a control signal. The discrete blocks are

**•**Block A, which has a sample time of 0.25 second

**•**Block B, which has a sample time of 0.5 second

**•**Block C, within the enabled subsystem, which has a sample time of 0.125 second

**•**Block D, also within the enabled subsystem, which has a sample time of 0.25 second

The enable control signal is generated by a Pulse Generator block, labeled Signal E, which changes from 0 to 1 at 0.375 second and returns to 0 at 0.875 second.



****

Blocks A and B execute independently of the enable control signal because they are not part of the enabled subsystem. When the enable control signal becomes positive, blocks C and D execute at their assigned sample rates until the enable control signal becomes zero again. Note that block C does not execute at 0.875 second when the enable control signal changes to zero.

**Triggered Subsystems**

Triggered subsystems are subsystems that execute each time a trigger event occurs.

A triggered subsystem has a single control input, called the *trigger input*, that determines whether the subsystem executes. You can choose from three types of trigger events to force a triggered subsystem to begin execution:

**•**rising triggers execution of the subsystem when the control signal rises from a negative or zero value to a positive value (or zero if the initial value is negative).

**•**falling triggers execution of the subsystem when the control signal falls from a positive or a zero value to a negative value (or zero if the initial value is positive).

**•**either triggers execution of the subsystem when the signal is either rising or falling.

For example, in the following timing diagram for a discrete system, a rising trigger (R) does not occur at time step 3 because the signal has remained at zero for only one-time step when the rise occurs.



A simple example of a triggered subsystem is illustrated.



In this example, the subsystem is triggered on the rising edge of the square wave trigger control signal.

**Creating a Triggered Subsystem**

You create a triggered subsystem by copying the Trigger block from the Signals & Systems library into a subsystem. Simulink adds a trigger symbol and a trigger control input port to the Subsystem block.



To select the trigger type, open the Trigger block dialog box and select one of the choices for the **Trigger type** parameter, as shown in the following dialog box:

****

Simulink uses different symbols on the Trigger and Subsystem blocks to indicate rising and falling triggers (or either). This figure shows the trigger symbols on Subsystem blocks.



**Outputs and States Between Trigger Events.** Unlike enabled subsystems, triggered subsystems always hold their outputs at the last value between triggering events. Also, triggered subsystems cannot reset their states when triggered; states of any discrete blocks are held between trigger events.

**Outputting the Trigger Control Signal.** An option on the Trigger block dialog box lets

you output the trigger control signal. To output the control signal, select the **Show output port** check box.



The **Output data type** field allows you to specify the data type of the output signal as auto, int8, or double. The auto option causes the data type of the output signal to be set to the data type (either int8 or double) of the port to which the signal is connected.

**Function-Call Subsystems**

You can create a triggered subsystem whose execution is determined by logic internal to an S-function instead of by the value of a signal. These subsystems are called *function-call subsystems*.

**Blocks That a Triggered Subsystem Can Contain**

Triggered systems execute only at specific times during a simulation. As a result, the only blocks that are suitable for use in a triggered subsystem are

**•**Blocks with inherited sample time, such as the Logical Operator block or the Gain block

**•**Discrete blocks having their sample times set to -1, which indicates that the sample time is inherited from the driving block.

**Triggered and Enabled Subsystems**

A third kind of conditionally executed subsystem combines both types of conditional execution. The behavior of this type of subsystem, called a *triggered* *and enabled* subsystem, is a combination of the enabled subsystem and the triggered subsystem, as shown by this flow diagram.

****

A triggered and enabled subsystem contains both an enable input port and a trigger input port. When the trigger event occurs, Simulink checks the enable input port to evaluate the enable control signal. If its value is greater than zero, Simulink executes the subsystem. If both inputs are vectors, the subsystem executes if at least one element of each vector is nonzero. The subsystem executes once at the time step at which the trigger event occurs.

**Creating a Triggered and Enabled Subsystem**

You create a triggered and enabled subsystem by dragging both the Enable and Trigger blocks from the Signals & Systems library into an existing subsystem. Simulink adds enable and trigger symbols and enable and trigger and enable control inputs to the Subsystem block.



You can set output values when a triggered and enabled subsystem is disabled as you would for an enabled subsystem.

**A Sample Triggered and Enabled Subsystem**

A simple example of a triggered and enabled subsystem is illustrated in the model below.



**Modeling with Control Flow Blocks**

The control flow blocks are used to implement the logic of the following C-like control flow statements in Simulink:

**•**for

**•**if-else

**•**switch

**•**while (includes while and do-while control flow statements)

Although all the preceding control flow statements are implementable in Stateflow, these blocks are intended to provide Simulink users with tools that meet their needs for simpler logical requirements.

**Creating Conditional Control Flow Statements**

You create C-like conditional control flow statements using ordinary subsystems and the following blocks from the Subsystems library.



**If-Else Control Flow Statements**

The following diagram depicts a generalized if-else control flow statement implementation in Simulink.



Construct a Simulink if-else control flow statement as follows:

**•**Provide data inputs to the If block for constructing if-else conditions. Inputs to the If block are set in the If block properties dialog. Internally, they are designated as u1, u2,..., un and are used to construct output conditions.

**•**Set output port if-else conditions for the If block.

Output ports for the If block are also set in its properties dialog. You use the input values u1, u2, ..., un to express conditions for the if, elseif, and else condition fields in the dialog. Of these, only the if field is required. You can enter multiple elseif conditions and select a check box to enable the else condition.

**•**Connect each condition output port to an Action subsystem.

Each if, elseif, and else condition output port on the If block is connected to a subsystem to be executed if the port’s case is true. You create these subsystems by placing an Action Port block in a subsystem. This creates an atomic Action subsystem with a port named Action, which you then connect to a condition on the If block. Once connected, the subsystem takes on the identity of the condition it is connected to and behaves like an enabled subsystem.

**Switch Control Flow Statements**

The following diagram depicts a generalized switch control flow statement implementation in Simulink.



Construct a Simulink switch control flow statement as follows:

**•**Provide a data input to the argument input of the Switch Case block. The input to the Switch Case block is the argument to the switch control flow statement. This value determines the appropriate case to execute. Noninteger inputs to this port are truncated.

**•**Add cases to the Switch Case block based on the numeric value of the argument input.

You add cases to the Switch Case block through the properties dialog of the Switch Case block. Cases can be single or multivalued. You can also add an optional default case, which is true if no other cases are true. Once added, these cases appear as output ports on the Switch Case block.

**•**Connect each Switch Case block case output port to an Action subsystem. Each case output of the Switch Case block is connected to a subsystem to be executed if the port’s case is true. You create these subsystems by placing an Action Port block in a subsystem. This creates an atomic subsystem with a port named Action, which you then connect to a condition on the Switch Case block. Once connected, the subsystem takes on the identity of the condition and behaves like an enabled subsystem. Place all the block programming executed for that case in this subsystem.

**Creating Iterator Control Flow Statements**

You create C-like iterator control flow statements using subsystems and the following blocks from the Subsystems library.



**While Control Flow Statements**

The following diagram depicts a generalized C-like while control flow statement implementation in Simulink.

In a Simulink while control flow statement, the While Iterator block iterates the contents of a While subsystem, an atomic subsystem. For each iteration of the While Iterator block, the block programming of the While subsystem executes one complete path through its blocks.

Construct a Simulink while control flow statement as follows:

**•**Place a While Iterator block in a subsystem.

The host subsystem becomes a while control flow statement as indicated by its new label, while {...}. These subsystems behave like triggered subsystems. This subsystem is host to the block programming you want to iterate with the While Iterator block.

**•**Provide a data input for the initial condition data input port of the While Iterator block.

The While Iterator block requires an initial condition data input (labeled IC) for its first iteration. This must originate outside the While subsystem. If this value is nonzero, the first iteration takes place.

**•**Provide data input for the conditions port of the While Iterator block. Conditions for the remaining iterations are passed to the data input port labeled cond. Input for this port must originate inside the While subsystem.



**For Control Flow Statements**

The following diagram depicts a generalized for control flow statement implementation in Simulink.



In a Simulink for control flow statement, the For Iterator block iterates the contents of a For Iterator Subsystem, an atomic subsystem. For each iteration of the For Iterator block, the block programming of the For Iterator Subsystem executes one complete path through its blocks.

Construct a Simulink for control flow statement as follows:

**•**Drag a For Iterator Subsystem block from the Library Browser or Library window into your model.

**•**You can set the For Iterator block to take external or internal input for the number of iterations it executes. Through the properties dialog of the For Iterator block you can set it to take

input for the number of iterations through the port labeled N. This input must come from outside the For Iterator Subsystem.

You can also set the number of iterations directly in the properties dialog.

**•**You can set the For Iterator block to output its iterator value for use in the block programming of the For Iterator Subsystem.

The iterator value is 1 for the first iteration and is incremented by 1 for each succeeding iteration.

The For Iterator block works well with the Assignment block to reassign values in a vector or matrix. This is demonstrated in the following example. Note the matrix dimensions in the data being passed.

