Developer's Guide To Pacemaker Development

Tutorial 1.3: Hello World

SFWRENG/MECHTRON 3K04
McMaster University

September 22, 2025

Getting Started With Modeling in Simulink®

"Hello World" is a time-honoured tradition in computer programming — a simple exercise that gets you started when learning something new. The aim of this tutorial to introduce Simulink programming by building a simple Simulink model. The background section demonstrates how models are represented in Simulink®. The tutorial section aims to challenge you to use Simulink® and Stateflow® to program your microcontroller to make the on-board LED blink.

Topics Covered

- How to implement a model in Simulink®
- Verifying a model in Simulink®.
- What is Stateflow®?
- Describing a system that toggles the on-board LED on the FRDM-K64F.

Prerequisites Complete the MATLAB® Simulink® installation procedure in accordance with the instructions from **Tutorial 1.2**

Additional Resources
Simulink Documentation
Simulink Model Basics
Stateflow Documentation
Stateflow Best Practises

1 Background

A model is an abstract description of a system. In Simulink®, a model is visually represented by a **block diagram**. Block diagrams in Simulink® are made up of two basic constructs: **blocks** and **lines**. Blocks perform specific functions and are used to generate, modify, combine, output and display signals. Lines connecting the blocks show the relationship between the components and are used to transfer signals from one block to another.

The Simulink Editor allows you to view and edit block diagrams by:

1. **adding blocks**,
2. **modifying blocks**,
3. **connecting blocks (with lines)**, and
4. **running simulations**.

The following demonstration illustrates how a model is represented in Simulink® by constructing an example block diagram for a simple model consisting of a sinusoidal input multiplied by a constant gain.

Demonstration — Building a System

The Simulink Editor is accessed by launching Simulink®

- from the MATLAB toolstrip by clicking the **Simulink** button, or
- from the MATLAB command prompt by entering the command `simulink`

and creating or opening a Simulink model.

1.1 Adding the Blocks

1.1.1 Adding Blocks Using the Library Browser

The **Library Browser** is used to navigate block libraries. The Library Browser can be opened by

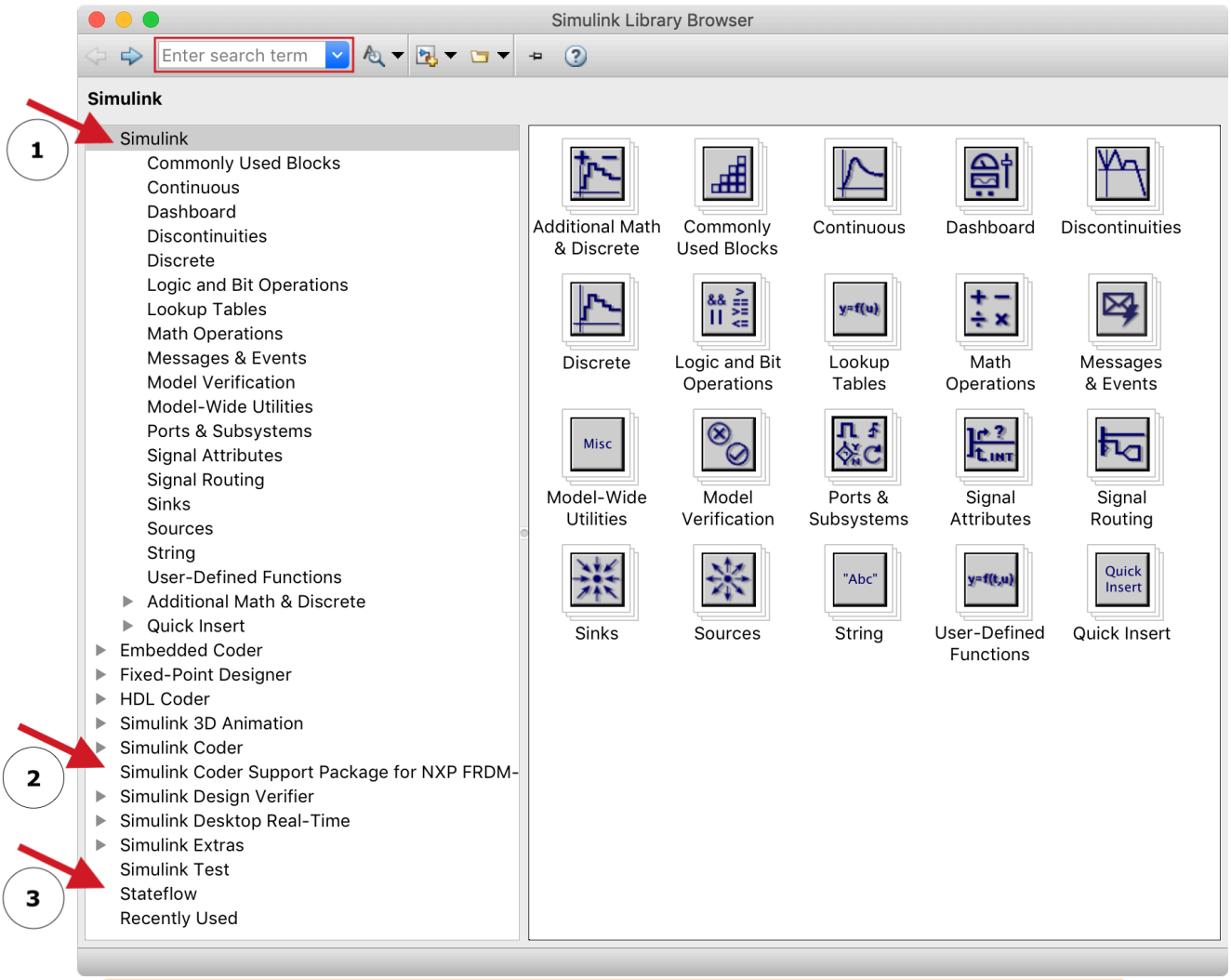
- clicking the **Library Browser** button in the **Simulation** tab.
- pressing CTRL + Shift + L for Windows, or ⌘ + Shift + L for Mac, on the keyboard.

Block libraries are organized by category in the left pane of the Library Browser. Each category is further decomposed into functional areas as shown in the figure below.

To locate a block,

- navigate to a category and/or functional area by clicking on a menu item on the left pane, *or*
- search for the block using the search field on the top left of the Library Browser window.

Then, drag a block from the right pane of the Library Browser to the Simulink Editor canvas to add it.



Tip: To set the Library Browser to stay on top of the other desktop windows, select the **Stay on top** button

A few block library categories that will be of particular interest to you in this course are:

1. the **Simulink** category,

   Simulink blocks provide a wide range of functionality. The most common functional areas are:

   - *Continuous* — comprises linear, continuous-time system elements such as integrators, transfer functions and state-space models.
   - *Discrete* — comprises linear, discrete-time system elements such as

integrators, transfer functions and state-space models.

- *Math Operations* — comprises mathematical operators such as gain, sum and dot product.
- *Ports & Subsystems* — comprises blocks related to subsystems, such as Inport, Outport, Subsystem and Model. A subsystem allows you to group a set of blocks together and establishes an interface with inputs and outputs.

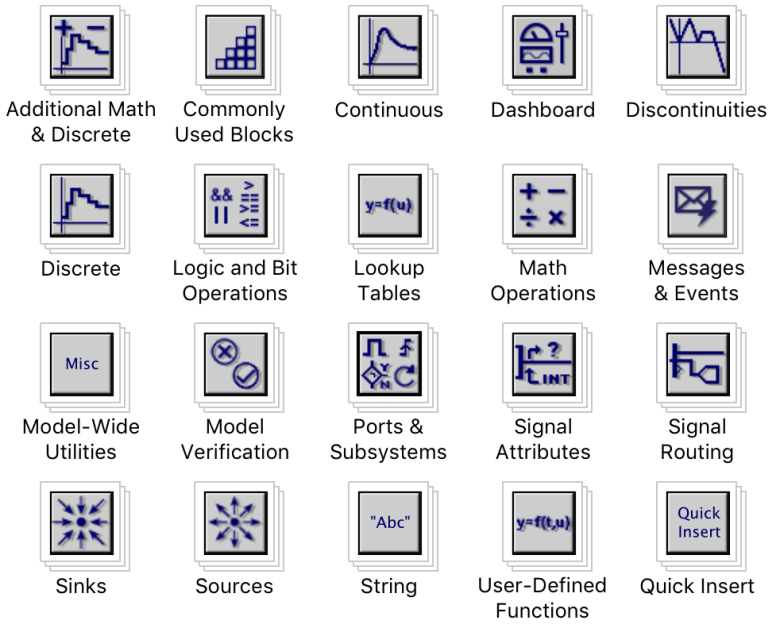


Figure 1: **Simulink category block library**

- *Sinks* — Used to output or display signals (using displays, scopes and graphs).
- *Sources* — Used to generate various signals such as step, ramp, square waves and sinusoidal signals.
- *User Defined Functions* — Used to define custom functions written in C or MATLAB.

② the **Simulink Coder Support Package for NXP FRDM-K64F Board** category

The Simulink Coder Support Package for NXP FRDM-K64F Board block library provides functionality to control the FRDM-K64F on-board peripherals.

Note: if you do not see this category in your Library Browser, make sure you have completed the Hardware Support Package installation procedure from **Tutorial 1.2 Part B**.

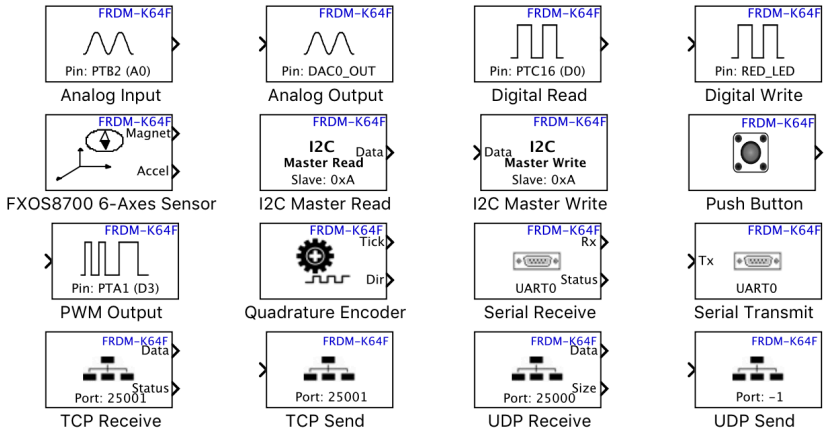


Figure 2: **Simulink Coder Support Package for NXP FRDM-K64F Board category block library**

③ the **Stateflow** category

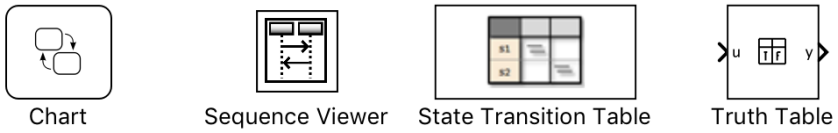
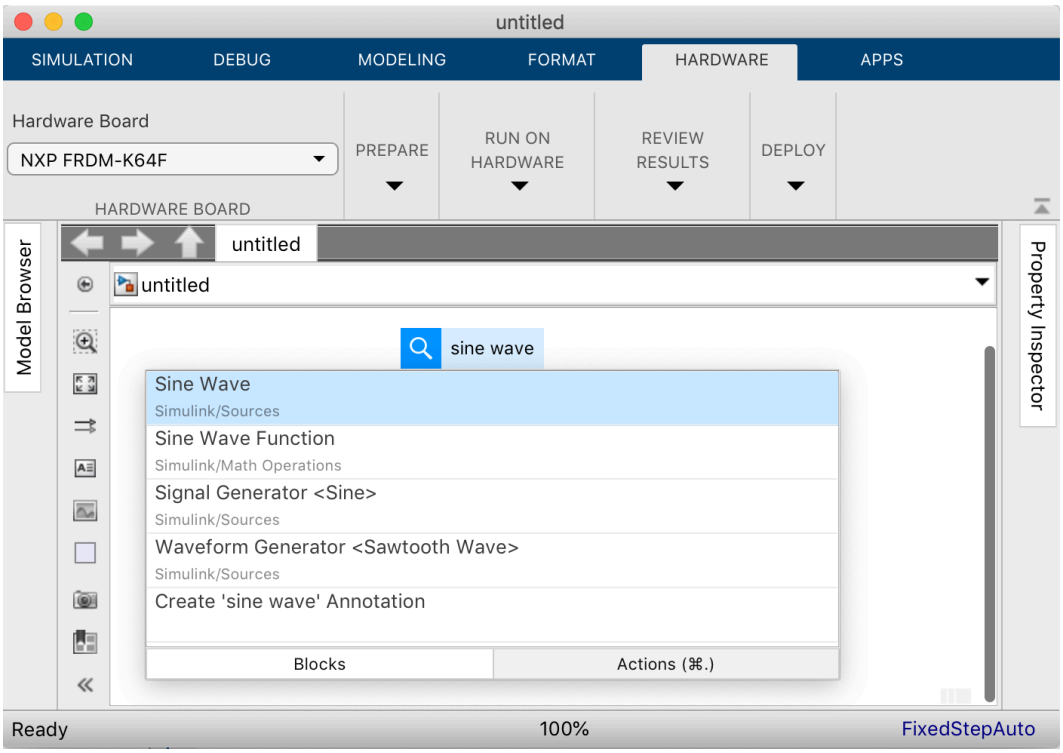


Figure 3: **Stateflow category block library**

The Stateflow block library provides a means of modeling combinational and sequential decision logic. You will most likely be using the Stateflow Chart during the project. More details on the Stateflow Chart is presented in the **Using Stateflow Charts** subsection below.

1.1.2 Adding Blocks Using the Simulink Search Feature

A quicker way to insert blocks to a block diagram is to click an empty area on the Simulink Editor and begin to type any part of the name of the block. The Simulink Editor will provide a drop down menu with several options that match the search criteria. When you see the name of the block you are looking for, you can use the ↑ and ↓ arrow keys on your keyboard to navigate to the block name and then press Enter. You can also double-click on the block name in the dropdown menu to insert the block.



The description under the block name indicates the category and functional area containing the block in the Library Browser. Note that some blocks from different libraries, such as the **"Sine Wave"** and **"Sine Wave Function"**, are equivalent because they can be modified to have the same functionality and visual representation. The Simulink Search Feature was used to obtain the **"Sine Wave"**, **"Gain"** and **"Scope"** blocks as shown in Figure 4.

1.2 Modifying the Blocks

1.2.1 Modifying the Block Name

Click a block once to select the block. The block name appears as a blue text when the default block name is being used. You can change the name of a block by

- clicking on the text that appears below the block.
- pressing F2 on the keyboard

Block names must be unique because the Simulink Editor relies on having block diagram elements with unique identifiers. When you drag another instance of the same block onto the canvas, Simulink® automatically suffixes the block name with an integer. If you attempt to rename a block with the same name as another block, the Simulink Editor will produce an error popup to alert you that a block has a duplicate name.

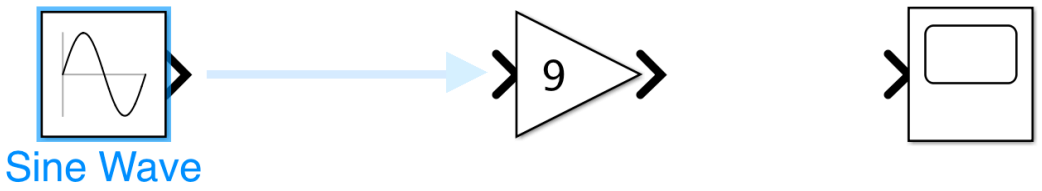


Figure 4: Block Diagram of the Sine Wave, Gain and Scope blocks

Tip: To fit all the blocks in your model to the size of the Simulink window, press Space. Space + F can be used to fit to selection. To pan across elements in your model, hold Space and click and drag your mouse.

1.2.2 Modifying the Block Parameters

Parameters of the block are modified to correspond to the system being modelled. Block parameters (and their initial settings) vary depending on the block. To modify the parameters, double-click on the block.

1. *Sine Wave Block Parameters*
   The following parameters are modified with their associated values:

   - amplitude: 2
   - frequency: π
   - phase: $\pi/2$

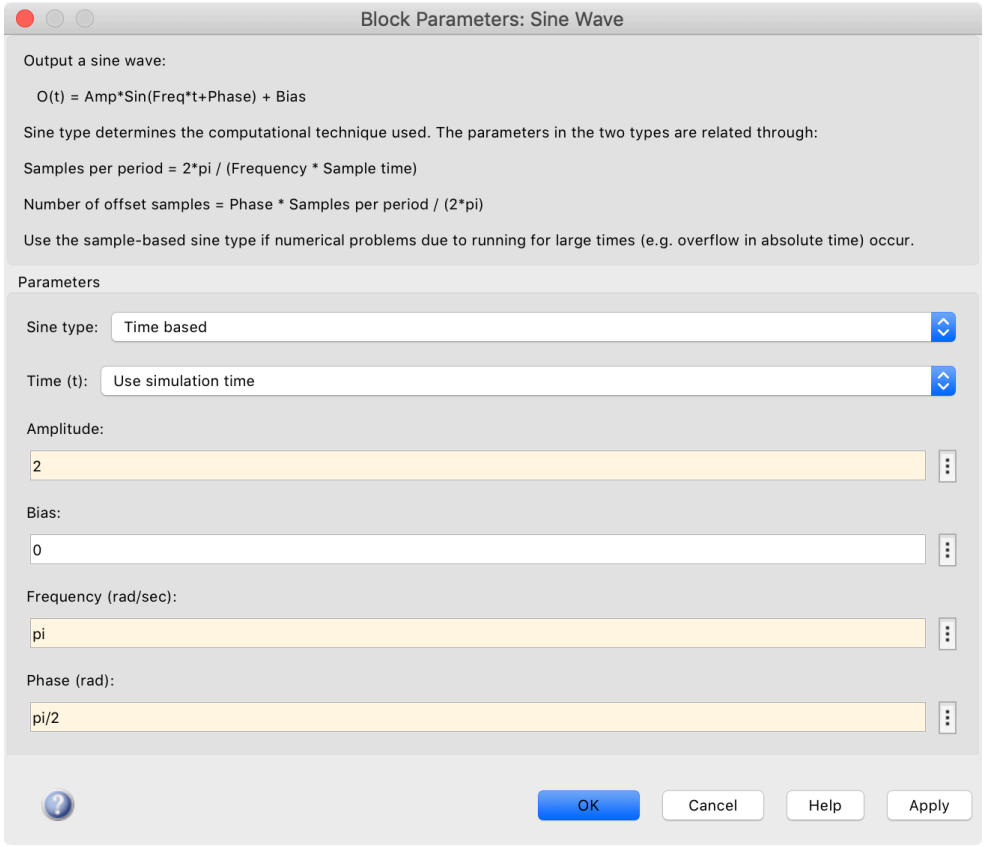


Figure 5: Block Parameters for Sine Wave

The highlighted text fields indicate that their values have been modified since the Block Parameters window was last open. Click "**OK**" or "**Apply**" to apply the changes.

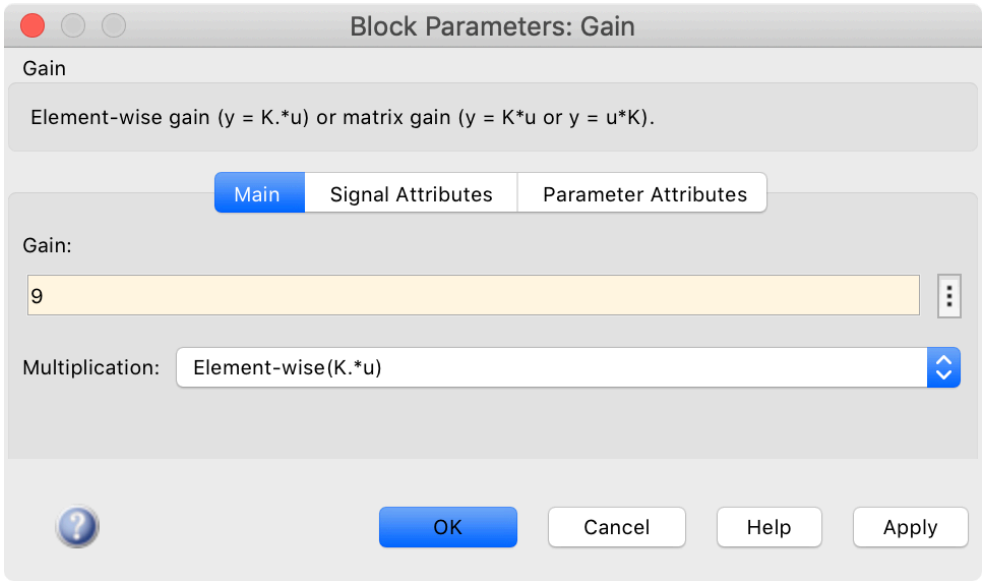
Note that the **Time(t)** parameter in Figure 5 is set to **Use simulation time**. The step size that you specified in the **Model Configuration Parameters** menu in Tutorial 1.2 is used to determine when the Simulink Engine performs a computation.

Suppose the model uses:

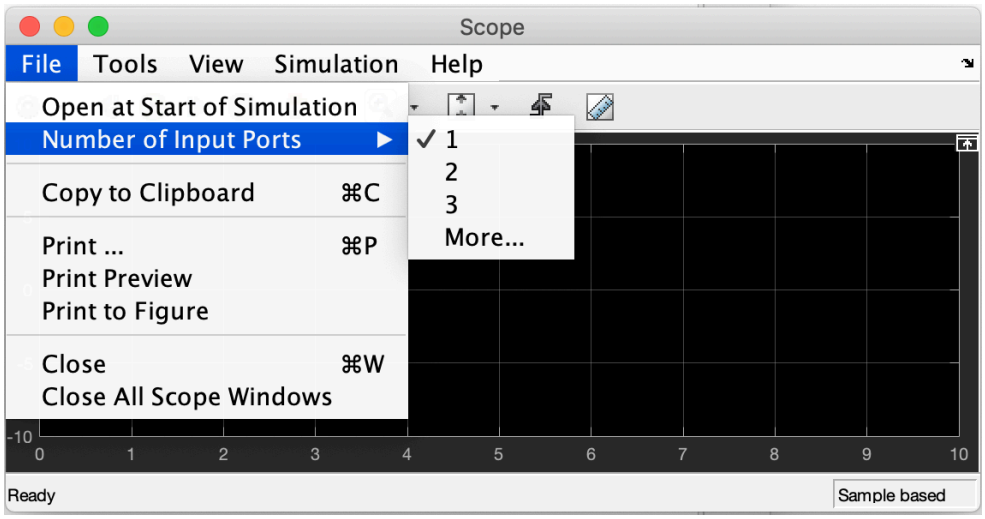
- fixed-step size: 0.001s
- Simulation stop time: 10.0s (default value)

The Sine Wave block effectively performs an element-wise operation on Time(t) input $\begin{bmatrix}0 & 0.001 & 0.002 & \dots & 9.999 & 10.000\end{bmatrix}$, where each element in the vector represents the simulation time at a time step.

2. *Gain Block Parameters*
   The **Gain** parameter is modified to a value of **9**. The input signal is a 1x1 vector so the Multiplication parameter is left at the default.

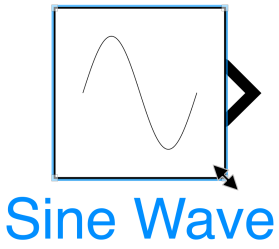


3. *Scope Block Parameters*
   The Scope block has several block parameters for changing the block's behavior, the block's interface and the layout of the display. The File menu of the block provides options to automatically **open the block at the start of simulation** and to **modify the number of input ports**. You may also right click anywhere on the grid to access the Configuration Properties and Style.



1.2.3 Modifying the Block Size

To perform a resize operation on a block, first click the block once to select it. Then, click one of the squares at the corners of the block and drag the mouse. To retain the aspect ratio, hold shift while dragging the mouse.



Tip: To zoom in, scroll up with your mouse or press CTRL + = for Windows (⌘ + = for Mac). To zoom out, scroll down with your mouse or press CTRL + - for Windows (⌘ + - for Mac).

1.3 Connecting the Blocks

The easiest way to create a connection is to **(1)** click a port, terminator, or line segment, and then **(2)** click a compatible, highlighted model element.

Alternatively, you can click and drag the mouse from the output terminal of a block to the input terminal (or data terminal) of another block.

To branch a signal (as shown in Figure 6):

- hold CTRL while clicking a point on a line; simultaneously drag the mouse from the line to another terminal, *or*
- perform a right click on the line while dragging the mouse from the line to the terminal.

A new input terminal will be automatically created as the line is dragged toward the left side of the Scope block.

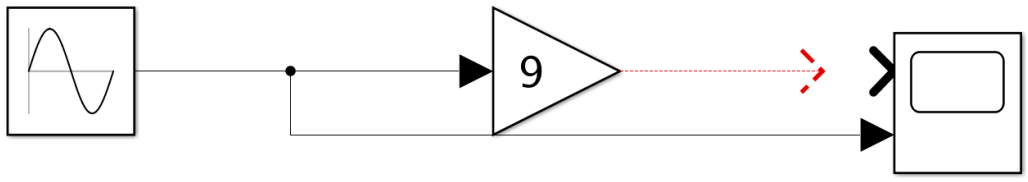
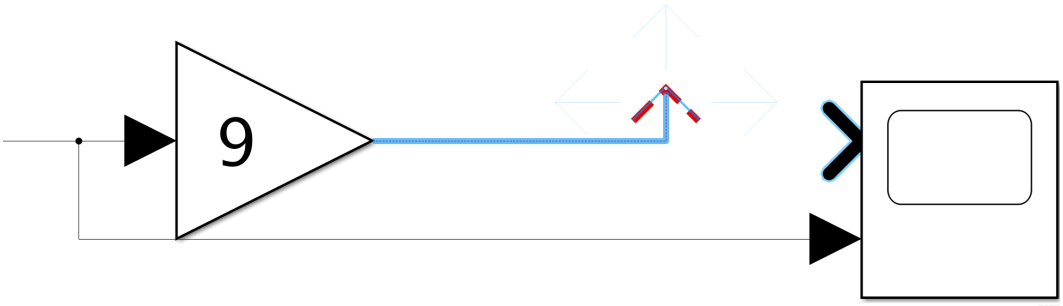


Figure 6: Complete and Incomplete Block Connections

The Simulink Editor will create a line with an arrow head that is filled in if a line is properly connected. If a line is not properly connected, the arrow head will not be filled in and the line will be a dotted red line as shown in Figure 6.

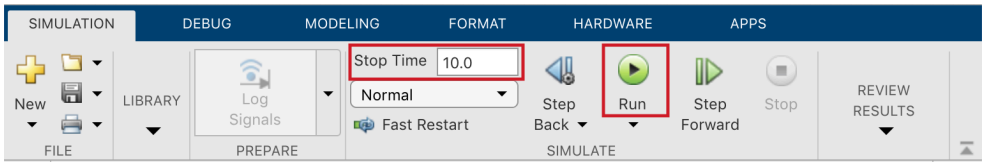
The Simulink Editor will attempt to provide suggestions to assist you connect blocks together more conveniently. When you move a block toward another block, the Simulink Editor may provide a blue arrow line between the output terminal of the Sine Wave block and the input terminal of the Gain block, as depicted in Figure 4 above. Click the blue line to connect the blocks together.



For more control over how lines are routed within a diagram, hover over a loose arrow tip and click the light blue arrows to guide the line to a terminal.

1.4 Running Simulations

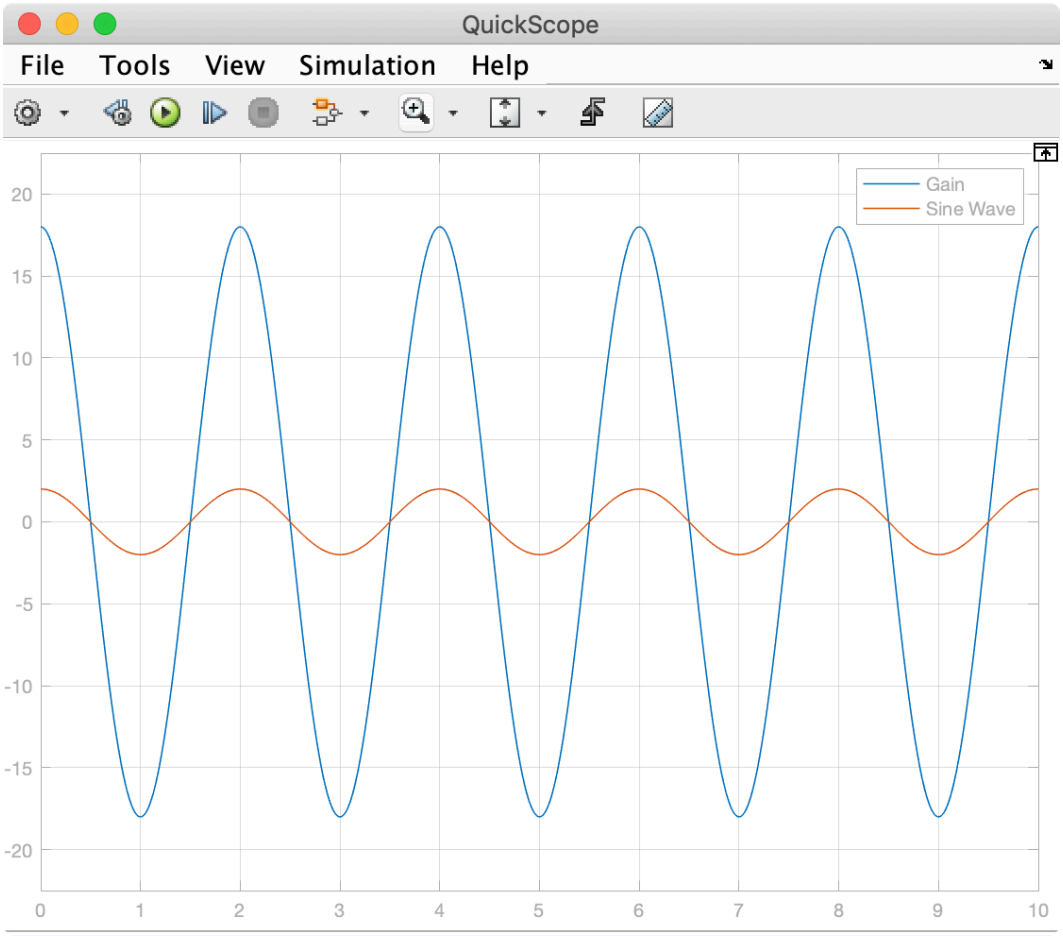
1. A simple way to verify your model is by running a simulation. To run a simulation, navigate to the **Simulation** menu in the top toolstrip and click the **Run** ▶ button.



   The **Stop Time** determines the length of the simulation. Recall our discussion on the **Fixed-step size** from **Tutorial 1.2**. For a 10.0 second simulation and a 0.001s step size, the simulation ends after 10000 iterations. The result of the simulation can be visualized after running a simulation by double clicking the scope block.

2. An alternate way of running a simulation is to double click the scope block and click the **Run** ▶ button from within the block's window.

The scope block allows us to visualize the output of the simulation by plotting the signal amplitude over simulated time. The output waveforms in the following figure appear to be correct. Since the original signal had a phase shift of $\pi/2$, it resembles a cosine function with an amplitude of 2. The amplitude of the amplified signal is 9 times the amplitude of the original signal.



Testing Tip: Incremental testing is very important in software engineering. It would be a good practise to verify your model by running a simulation each time you add a new feature. This can be achieved by branching the signals to a scope block and verifying the output.

Impact of the Step Size on the Simulation

Consider the impact of the step size on the simulation. How would the above plot change if:

1. a variable step size is used instead of a fixed-step size?
2. a fixed-step size of 0.2s is used instead of 0.001s?

Rationale for not using a variable step solver scheme: A solver applies a numerical method to solve a set of ordinary differential equations representing the model. During each computation, the solver determines the time of the

next simulation step. When a variable step size solver is used, the step size is increased when the signal changes slowly and decreased when when the signal changes quickly, to minimize the total number of time steps while maintaining a specified level of accuracy. Since we will be developing models for deployment to a real-time computer system, the step sizes will need to be mapped to the real-time clock on the FRDM-K64F hardware. Variable-step sizes cannot be mapped to the real-time clock, so the model should always be configured with a **fixed-step size** as noted in **Tutorial 1.2 Part C**.

Step size magnitude: Generally, smaller step sizes are preferred because they improve the signal resolution. For our purposes, we are using a step size of 1 ms. Although the FRDM-K64F hardware is capable of performing computations at step sizes that are smaller than 1 ms, the 1 ms step size value was chosen due to a limitation of the MATLAB® engine.

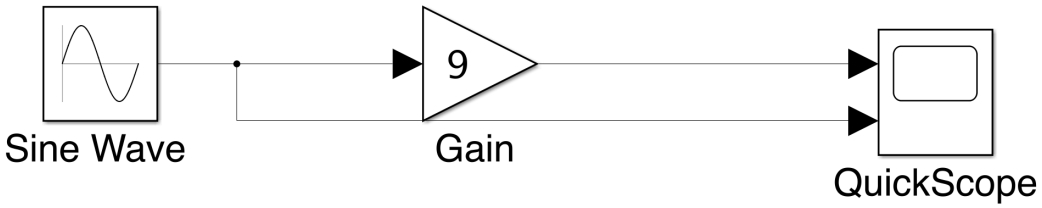
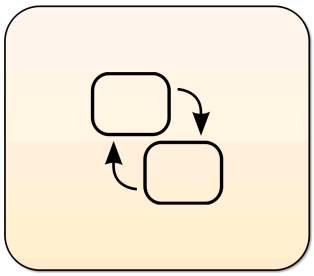


Figure 7: Block Diagram of the Simple Model

Using Stateflow Charts — The Basics

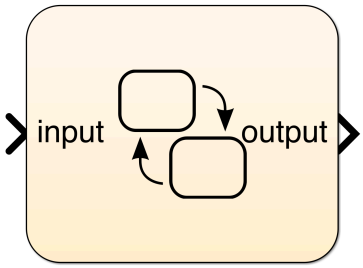
Basic Simulink blocks are best suited for modeling functional/mathematical algorithms and control processes that need to respond to continuous changes. Stateflow® extends Simulink® with options to implement state charts. Stateflow charts allow us to practically model reactive systems that require more logical and relational operators and need to respond to instantaneous changes. An example of such a system is one that is based on **finite-state machines (FSMs)**.



A Stateflow chart is visually represented on a block diagram as a block with an image of two states with transitions between them.

Stateflow Chart Interface

Stateflow charts allow you to establish an interface to enable connections between the chart and other blocks in the block diagram for signal and data transfer. A "closed-loop" Stateflow chart has an output that is connected back to the input to implement a self-correcting logic. An "open-loop" chart does not have feedback.



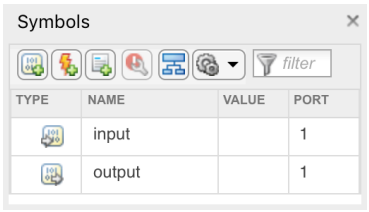
There are several ways to create data variables that specify chart inputs and outputs. After adding the inputs and outputs, the Stateflow chart block should have terminals at its edges.

A. *Using the Stateflow menu: (for quick additions)*

1. Double-click the chart to enter the chart.
2. Right-click the canvas.
3. Select **Add Inputs & Outputs** > **Data Input From Simulink** for inputs or **Add Inputs & Outputs** > **Data Output To Simulink** for outputs.
4. Modify the variable properties accordingly (refer to Basic Stateflow Chart Elements — Data).

B. *Using the Symbols Pane: (most convenient)*

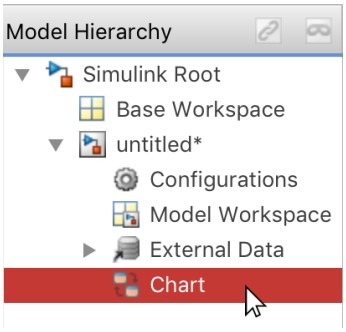
1. Double-click the chart to enter the chart.
2. Under the **Modeling** tab in the top toolstrip, click the **Symbols Pane**  icon.
3. Click the **Create Data**  icon in the Symbols Pane to add a new variable. The new variable will be represented as a new row in the Symbols Pane.
4. Click the icon under the **TYPE** column along the variable's row. Change the data type to Input or Output.
5. Change the port number by entering a number along the **PORT** column.



6. Right click the row corresponding to the variable and select **Inspect...** to modify the variable properties (refer to Basic Stateflow Chart Elements — Data).

C. *Using the Model Explorer: (for accessing a wider range of variables across the model)*

1. Open the Model Explorer:
   - Click the **Model Explorer** icon on the **Modeling** tab in the top toolstrip, *or*
   - Right click any where on the block diagram canvas and select "**Explore**", *or*
   - Press CTRL + H on the keyboard
2. Create a new variable
   1. Make sure the Chart is selected in the **Model Hierarchy** pane on the left hand side of the Model Explorer window since these variables are to exist in the scope of the chart.



   2. Click the **Add Data** icon on the top toolbar.
3. Modify the properties of the new variable (refer to Basic Stateflow Chart Elements — Data).
   - make sure the Dialog View is enabled and then click the variable in the Contents window. The variable properties will show up on the right hand side of the Model Explorer Window, *or*
   - perform a right click on the created variable in the Contents window and then click "**Properties...**"

Stateflow Chart Basic Elements

Stateflow charts are made up of three main constructs: states, transitions and data.

States

States can be added to the diagram by clicking the **State** icon on the left pane and then clicking an area of the canvas to place the state. Alternatively, a drag and drop operation from the left pane to the canvas can be used to add the state.

Text-based coding is used within Stateflow charts to specify state actions within states. The following figure illustrates the notation of a basic format for writing code in states. The items within the angle brackets <> (inclusive) are placeholders for the specified information. When creating your model, replace the items within the angle brackets <> (including the brackets) with valid code as per the discussion below. For example, replace <identifier> with the actual name of the state. (**Note:** the tutorial section provides an example implementation following this format).

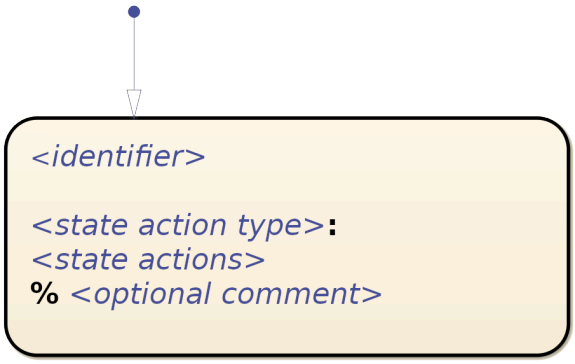


Figure 8: Stateflow Notation — Basic Format for State Programming

The code in a state block must begin with a valid C **identifier** to name the state. Then, one or more **state action types** may be specified for the state: *entry*, *during*, *exit*, etc. State action types determine when the state actions following it is executed. In most situations only the entry state action type is required. State action types must be followed by a colon. **State actions** are statements written in C or MATLAB. State actions are written directly below the declaration of its corresponding state action type.

State transitions

Transitions represent the system passing between states. Transitions are represented by blue lines connecting chart objects together.

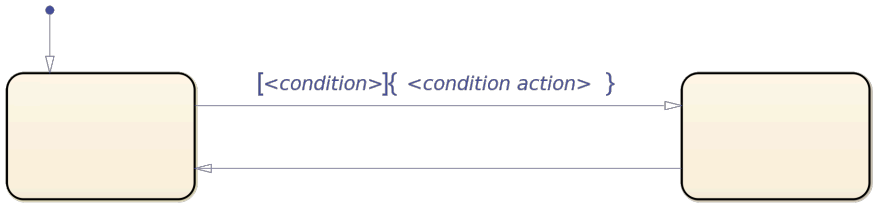


Figure 9: Stateflow Notation — Basic Format for Transition Labels

Text-based coding is used to specify transition logic on transition lines. Labels on the transitions are written code that specify conditions and/or actions for the transition. Conditions are generally written between square parentheses `[]`. Basic conditions are based on

- decision-based logic operations (i.e. `[my_input < 0]`)
- temporal (time-based) logic operations (i.e. `after(n,sec)`, where `n` represents the magnitude of a time delay and `sec` represents the time unit)

If no condition is specified, the transition is an unconditional transition. Sometimes conditions are followed by expressions enclosed in curly braces `{}` to specify condition actions that are to be executed during the transition when the condition is satisfied.

A default transition determines the initial state the model enters at the start of execution. Default transitions have a solid dot and a line with an arrowhead emanating from it. Stateflow charts and subcharts that have at least one state must have a default transition transition connected to a state.

For more information on conditions, condition actions and transitions, refer to Transitions.

Data

Data represents all variables that are referenced by the chart, including the interface variables (input and output). Variables are defined following a similar process as the one described above for defining the chart interface. Sometimes data, enumerations, classes and functions can also be transferred between MATLAB®, Simulink® and Stateflow®.

When defining data, it is important to specify the variable properties correctly. A common error you might run into when building the model is a data type propagation error that occurs when the **Type** field in the Data Properties is not consistent across all blocks that reference the signal that is mapped to the variable. By default, the **Type** parameter is set to "Inherit: Same as Simulink", which means the chart will assume the type specified by a preceding blocks that reference the signal, or *double* if no type was specified by any block.

It is generally a good idea to explicitly specify the data type rather than use "Inherit: Same as Simulink" and to follow the signal line across the model to make sure there are no data type mismatches. For more information on data properties, refer to `https://www.mathworks.com/help/stateflow/ug/set-data-properties-1.html`.

Advanced Stateflow Chart Objects

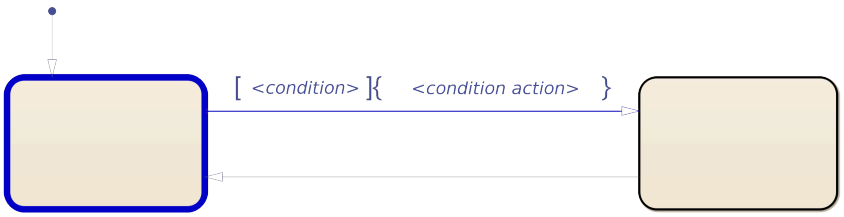
Charts allow us to represent hierarchy, parallelism and history. You can create superstates by nesting states within states, you can run states simultaneously by using parallel states, and you can use a History Junction to determine the active substate within a superstate based on past decisions.

Additional chart objects available include: junction, Simulink and MATLAB functions, state transition tables, truth tables, atomic subcharts, messages and events. A common mistake is to attempt to use junctions as empty states. Junctions are not states — it is not possible for the system to enter a junction. Junctions effectively act as decision structures for all conditions along a transition path. For more information on how Stateflow® evaluates transitions, refer to `https://www.mathworks.com/help/stateflow/ug/evaluate-transitions.html`.

Design Tip: Take some time get familiar with different kinds of blocks and software tools available in Simulink® and Stateflow®. Try investigating other Library Browser blocks and Stateflow chart objects. Refer to the Simulink and Stateflow documentation. When designing pacemaker model, assess the benefits and tradeoffs of using each the tool and whether you can leverage the use of the components to apply the software design principles.

Stateflow Simulation

Set the Simulation Stop Time to **Inf**. Verify the model by pressing the **Run** ▶ button in the **Modeling** ribbon on the toolstrip. You can verify the model by observing the system stepping through the states (the active state will be highlighted with a thick blue line).

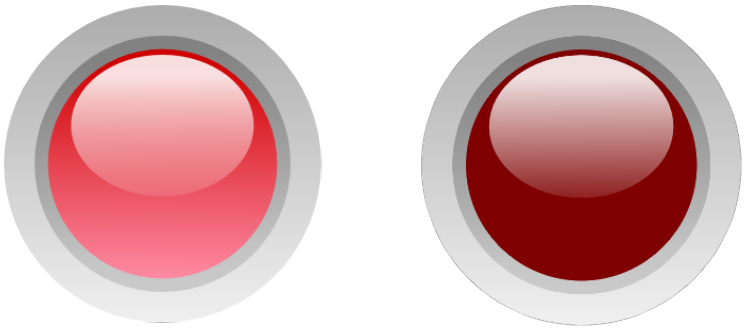


Refer to the Debug ribbon in the toolstrip for other tools for debugging your model. To use breakpoints to pause your model during simulation when a state is entered, right click a state and select **"Add Breakpoint"** before running the simulation.

2 Tutorial

Blinking LED Implementation

In this lab you will use Simulink® to program your microcontroller to make an LED blink. A blinking LED can be modelled as an FSM with two states. You will use Stateflow® to implement the FSM.



STATE 1: LED_ON STATE 2: LED_OFF

Create a New Model

Create a new model using the template you created in **Tutorial 1.2**.

Create a New Chart with Inputs and Outputs

Click in the Simulink model window and search for "Chart" to add a new chart to your Simulink model. This is where you will add the logic to implement your program. You will need to configure the inputs and outputs for the chart.

For this chart, the inputs will be the amount of time the light should stay off between blinks, the amount of time the light should blink for. The output will be the LED state (on or off). The chart below describes the correct setup of these variables.

Variable	Type	I/O
DURATION_ON	Numerical type (example: int16)	Input
DURATION_OFF	Numerical type (example: int16)	Input
LED_STATE	Boolean	Output

The output variable needs to be mapped to the appropriate pin on the microcontroller so that it can actually be used to control the LED light. Do this by clicking in the Simulink model window and searching for the "Digital Write" block for the K64F. Select an LED pin to write this value to. Connect

your output variable from your chart to the digital write block by clicking and dragging the output over to the digital write block.

Create States in the Chart

Double-click the chart you've created in the Simulink model window to open it. In this program, since the light can either be on or off, only 2 states are required to implement the desired functionality. Add a state by dragging it into the workspace from the toolbar on the left side, and give your state a descriptive name (for example, LED_ON or LED_OFF). You will need one state for when the LED is on and one state for when the LED is off, so create both now.

One of the states you set up will need to be defined as the starting state. In this program it is not very important which state is the starting state, but in other applications it may be important to make sure the program functions correctly. Do this by dragging a "default transition" from the toolbar on the left to the state that you want to be the starting state.

Create State Transitions

In order to get the light to actually blink, you need to set up transitions between the states in your chart. If no transitions are set up, the program will just remain in the starting state indefinitely. Transitions occur between states when their conditions are met. In this program, the transitions will be required to turn the LED on, and to turn the LED off. The table below summarizes the transitions you will need to set up.

Starting State	Ending State	Condition
LED_ON	LED_OFF	after(DURATION_ON, msec)
LED_OFF	LED_ON	after(DURATION_OFF, msec)

Click and drag from the border of the starting state to create a transition to the ending state (drag to the ending state). Click the transition arrow to modify the condition of the transition. When programming the condition you have access to the chart variables you defined earlier.

Create State Entry Procedures

You have set up your chart with input and output variables, states, and state transitions, but you still need to set up the logic for those variables to determine what happens when state transitions occur, or when the program is in a particular state. Procedures can be defined for "entry", "during", and "after" for each state. In this program, you only need to use the "entry" state. Click

on the body of the state in your chart to define these procedures. Here you can modify variables within the scope of the chart.

The only thing you need to do in the entry procedure for each state in this program is to flip the LED_STATE variable to true or false to control the LED pin on the microcontroller. Your final state should look something like the box below when you're finished. You will need to do this for both states so the LED can turn on and off when the program enters the appropriate state.

```
LED_ON
entry:
LED_STATE = true;
```

Tip: Hold CTRL and click and drag a state to make a copy. Sometimes Stateflow® will automatically come up with a reasonable name for the new state.

Define Constants to Connect with Chart Inputs

Your chart has inputs set up for the durations of the on and off portions of the blink, but no value is yet being assigned to those inputs. To set values for each of the inputs, go to the Simulink model window and click in the workspace, then search for "Constant" and add a constant block. Give your constant block a name, and set the value by clicking within the block. Click and drag the constant block output to the appropriate chart input to feed the constant value to the input. Do this for both duration inputs.

You also need to set up the value type for the constants. To do this, double-click the constant block and look in the signal attributes tab to select the type. Similar to a typed programming language, the type of the constant needs to match the type of the input you're connecting it with. If you selected **int16** as the type for your chart inputs, select **int16** here as well.

Test the Output Using a Scope

Programming the microcontroller to run your program on the hardware can take some time. If you want to test your program using a Simulink to simulate the output instead, you can do this using a Scope block. Click on the workspace in the Simulink model window and search for a "Scope" block, and add this block. Connect the chart output to the Scope block.

Set the length of time for the simulation to run for (for example, 10 seconds),

and run the simulation. Double-click the scope to see the changes to the output variable over time. If the output isn't what you'd expect, go back and make sure you haven't made a mistake.

Save the Model

Save the model on your computer. The next tutorial will introduce how to deploy the program to the microcontroller and verify the model externally.

If you have received your kit and would like to attempt to run the program on hardware:

Use the "Deploy to hardware" button on the Hardware tab in the top toolstrip to compile the program and flash the board. This can take some time. Once the flashing is complete you should see the LED blinking on your microcontroller.

If you do not see the Hardware tab, make sure you have set up the Model Configurations in accordance with the instructions from **Tutorial 1.2 Part C**.

If you are having issues, please visit the troubleshooting section.

Challenge Exercise

The on-board LED on the FRDM-K64F board is a multicolor RGB LED. See if you can implement a switching feature that allows you to cycle between the following modes when a push button on the FRDM-K64F is pressed:

1. same functionality as above
2. same functionality as above, except the LED toggles to a different colour instead of being turned off in the LED_OFF state
3. the LED is off all the time

3 Troubleshooting

3.1 Board Firmware Update

If you encounter a problem where the FRDM-K64F microcontroller refuses to flash or always enters the bootloader or maintenance mode (i.e. the board shows up as the "BOOTLOADER" when it is connected to your computer), you may need to reflash the board firmware. (Recall the discussion of **bootloader** from **Tutorial 1.1**).

To replace the firmware on the MCU with the Segger J-Link OpenSDA firmware, connect one end of your micro-USB cable to the **OpenSDA USB** input on the FRDM-K64F board and the other end to the USB input on your system. The right side mirco-USB port on the FRDM microcontroller should be plugged in for power. Follow instructions on the webpage **"Install Segger J-Link Firmware"** to update the FRDM-K64F board with the Segger J-Link firmware. If you are using a Mac system, refer to the note below.

After the firmware has been updated successfully, the OpenSDA LED by the reset button will be solid red. The LED will stay on until you flash a program to the board.

Note: Due to limitations of the native Mac file explorer, replacing the firmware to use it as a J-Link cannot be done on Mac systems using the drag and drop operation. If you are using a Mac system, you will need to flash the FRDM-K64F using the command line interface. Replace Step 4 and Step 5 in the Segger J-Link Firmware update instructions with following steps:

1. Open a **Terminal** window (you can use Spotlight Search to find Terminal).
2. Double click the the following command to select it. Then copy and paste the command into a text editor of your choice.

   sudo mount -u -w -o sync /Volumes/BOOTLOADER; cp -X <path to interface firmware file> /Volumes/BOOTLOADER/

3. Replace the following placeholder *(angle brackets inclusive)* from the command above

   <path to interface firmware file>

   with the **full path** to the *02_OpenSDA_FRDM-K64F.bin* binary on

your system. Copy and paste the modified command into the Terminal window.

For example, if your binary is on your desktop, then your command would be

```
sudo mount −u −w −o sync /Volumes/BOOTLOADER; cp −X
    ~/Desktop/02_OpenSDA_FRDM−K64F.bin
    /Volumes/BOOTLOADER/
```

Note If you are having trouble with determining the path, you can move the binary to your desktop and copy and paste the command here in Step 3 verbatim. Alternatively, you can simply copy and paste first part of the command `sudo mount -u -w -o sync /Volumes/BOOTLOADER; cp -X` into the Terminal window. Then *click and drag the binary file to the Terminal window*. Then copy and paste the `/Volumes/BOOTLOADER/` part of the command.

4. Press Enter to run the command.

4 Revision History

Version	Date	Modification	Modified by
1.0	Sept. 16, 2017	Initial SE 3K04 Lab 1 document	Ayesh, Mostafa
2.0	Sept. 24, 2020	Refined content; new template	Kehinde, Michael
3.0	Sept. 2, 2025	Updated wording	Zavaleta, Angela
3.1	Sept. 22, 2025	Moved Troubleshooting Guide	Kapustin, Vasily