Modeling Patterns for C Constructs

A guide for generating popular C constructs from models using Real-Time Workshop® Embedded Coder™

Summary

- **This presentation is a reference guide for transitioning from hand coding in C to modeling with automatic code generation**
- **For a typical C pattern:**
  - Different modeling methodologies are illustrated using Simulink®, Stateflow®, and Embedded MATLAB™
  - Code from Real-Time-Workshop® Embedded Coder™ is shown
  - The mapping of the C construct to the model and its generated code is illustrated
  - All the steps involved in creating the model and generating the code are detailed in the notes page for each slide

- **Readers are encouraged to request or submit other patterns for future releases of this document by contacting:**
  - **arvind.jayaraman@mathworks.com or**
  - **tom.erkkinen@mathworks.com**

Caveats

- This document does not cover every modeling method or C construct
- MathWorks Release R2008a is used and results may differ for other versions
- White-spaces in the generated code snippets have been changed in some cases for clarity
- Working knowledge of MATLAB and Simulink is helpful for readers but not required

Additional Resources

- **C/C++ Code Generation Web Videos**
  - **www.mathworks.com/products/rtwembedded/demos.html**
- **HDL Code Generation Web Videos**
  - **www.mathworks.com/products/slhdlcoder/demos.html**
- **Fixed Point Code Generation Web Videos**
  - **www.mathworks.com/products/simfixed/demos.html**
- **Fixed Point Modeling and Code Generation Tips**
  - **www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=19835**

Index

- Standard Steps to prepare models
- Types, Operators and Expressions
- Control Flow
- Functions and Program Structure
- Structures
- Arrays and Pointers

Types, Operators and Expressions

C Pattern
Data Declaration
Data-Type Conversion
Type Qualifiers
Relational and Logical Operations
Bitwise Operations

Return to Table of Contents

Control Flow

C Pattern
If-Then-Else
Switch-Case
For-Loop
While-Loop
Do-While Loop

Return to Table of Contents

Functions & Program Structure

C Pattern
Functions ▪void-void Functions ▪Functions returning arguments ▪Calling external C functions
#define

Return to Table of Contents

Structures

C Pattern
Typedef
Structures for Parameters
Structures for Signals
Nested Structures
Bit Fields

Return to Table of Contents

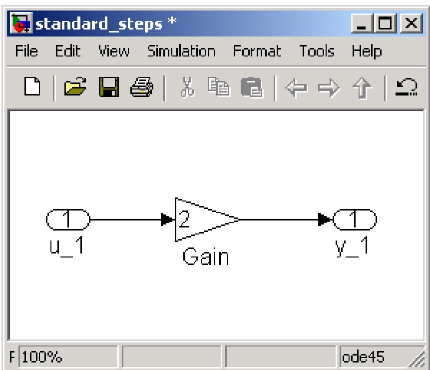
Arrays and Pointers

C Pattern
Arrays
Pointers

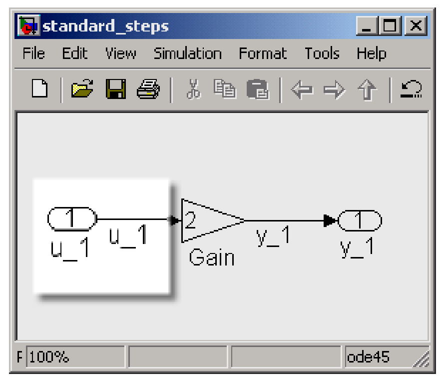
Return to Table of Contents

Standard Steps adopted in this document to prepare models for Code Generation

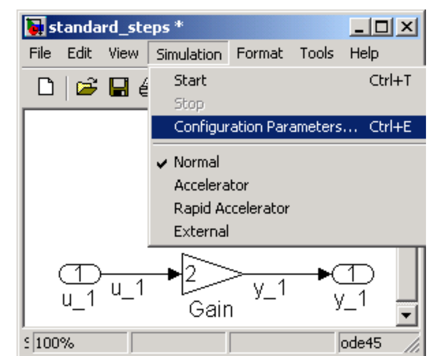
Create the model



Label Input/Output Signals as shown

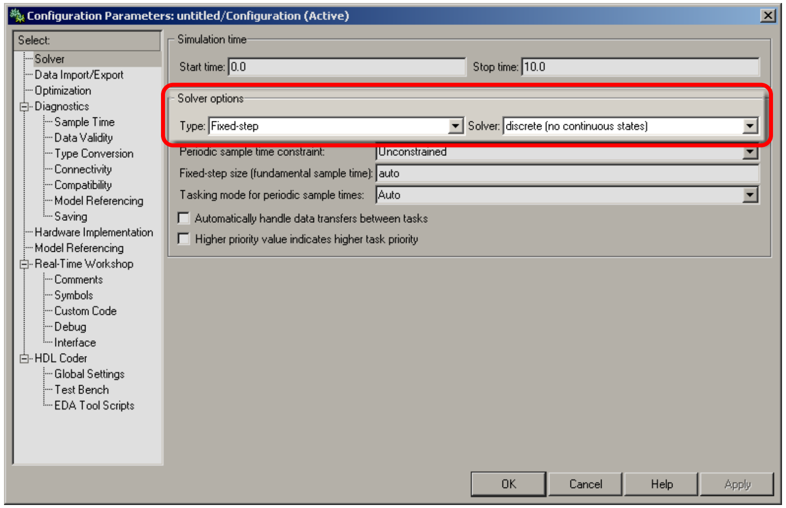


From the file menu choose ‘Simulation’ and select ‘Configuration Parameters’

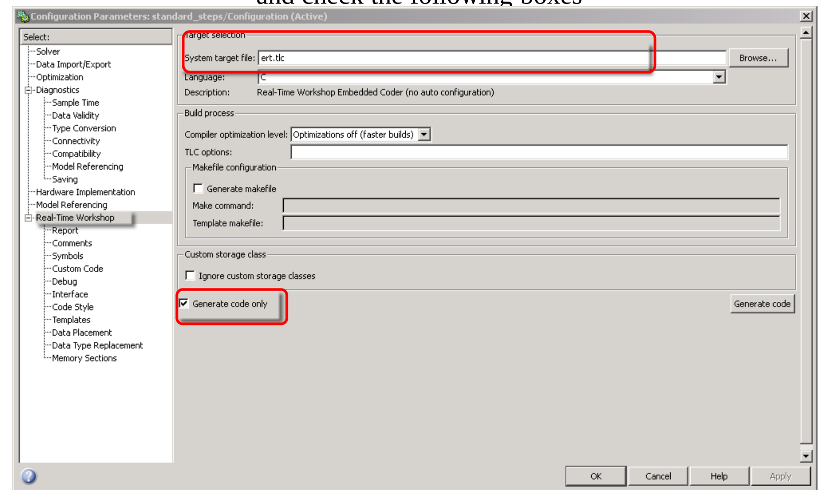


Select Configuration Parameters

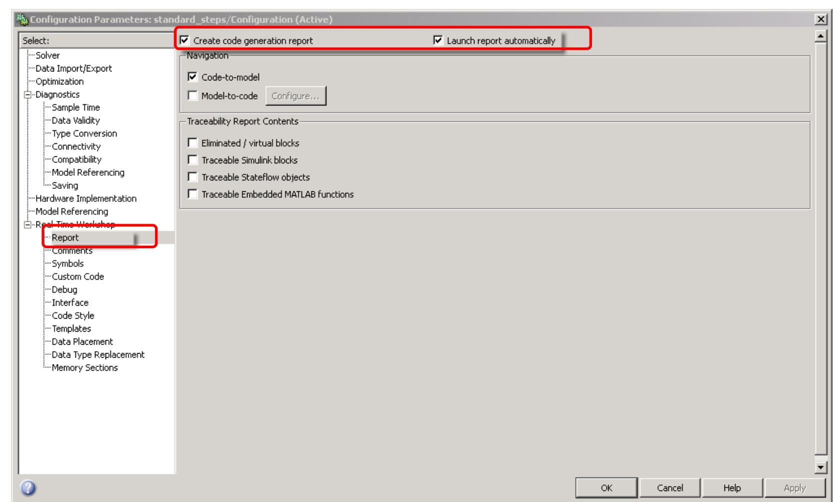
From the Solver Pane: Select Fixed Step discrete Solver



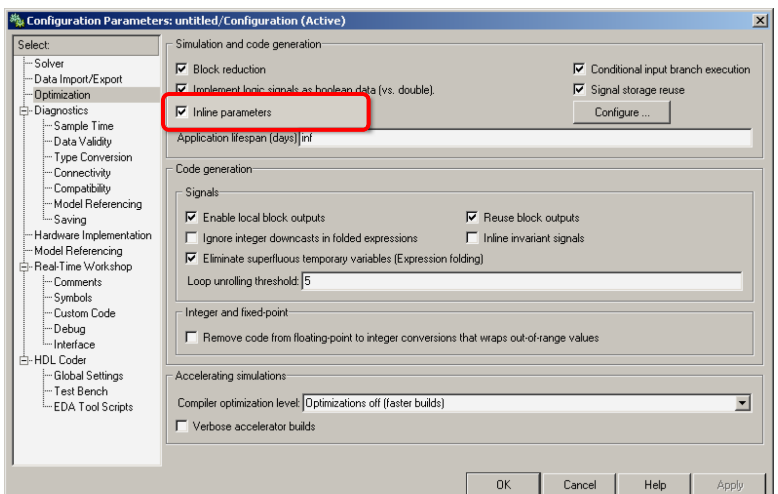
In the Real-Time Workshop Pane: Choose ert.tlc as System Target File and check the following boxes



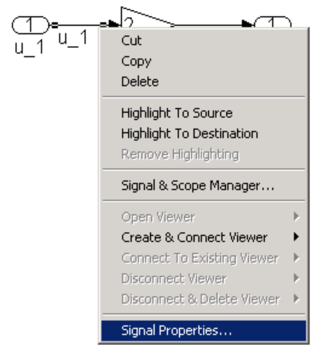
Enable code generation Report



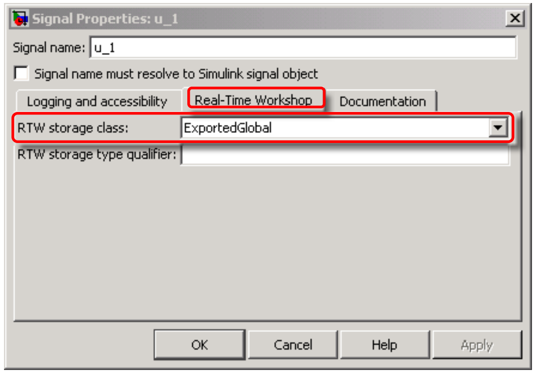
In the Optimization Pane: Select Inline Parameters



Right Click on Signal Lines and
Select "Signal Properties"



In the Real Time Workshop pane select the RTW storage class
to be Exported Global



We are now ready to generate code !

Return to Table of Contents

Glossary

- **CSC** : Custom Storage Class
- **Signal** : An entity whose value is expected to change during simulation. Generally a connecting wire that carries data is referred to as a ‘signal’.
- **Parameter** : An entity whose value typically remains constant during simulation run-time. Constants, gains etc. are typical examples of parameters.
- **Simulink Data Object**: An entity that specify values, data types, tunability, value ranges, and other key attributes of block outputs and parameters.
- **Virtual Subsystem**: Virtual subsystems visually organize blocks but have no effect on the model's functional behavior.
- **Non-Virtual Subsystem**: A subsystem whose algorithm is executed atomically.
- **Non-Virtual Bus** : A Bus whose signals are copied/packed into a data-structure.

C Code Pattern: Data Declaration

Code Pattern, Parameter	Code Pattern, Signal
int p_1 = 3 ; Description: *The aim is to obtain declarations such as the ones above in the generated code. p_1 is a parameter that has a 32 bit integer data-type.*	int u_1 ; Description: *u_1 is a signal that has an integer data-type. In this example the size of the data type int is 16 bits.*

Return to Table of Contents

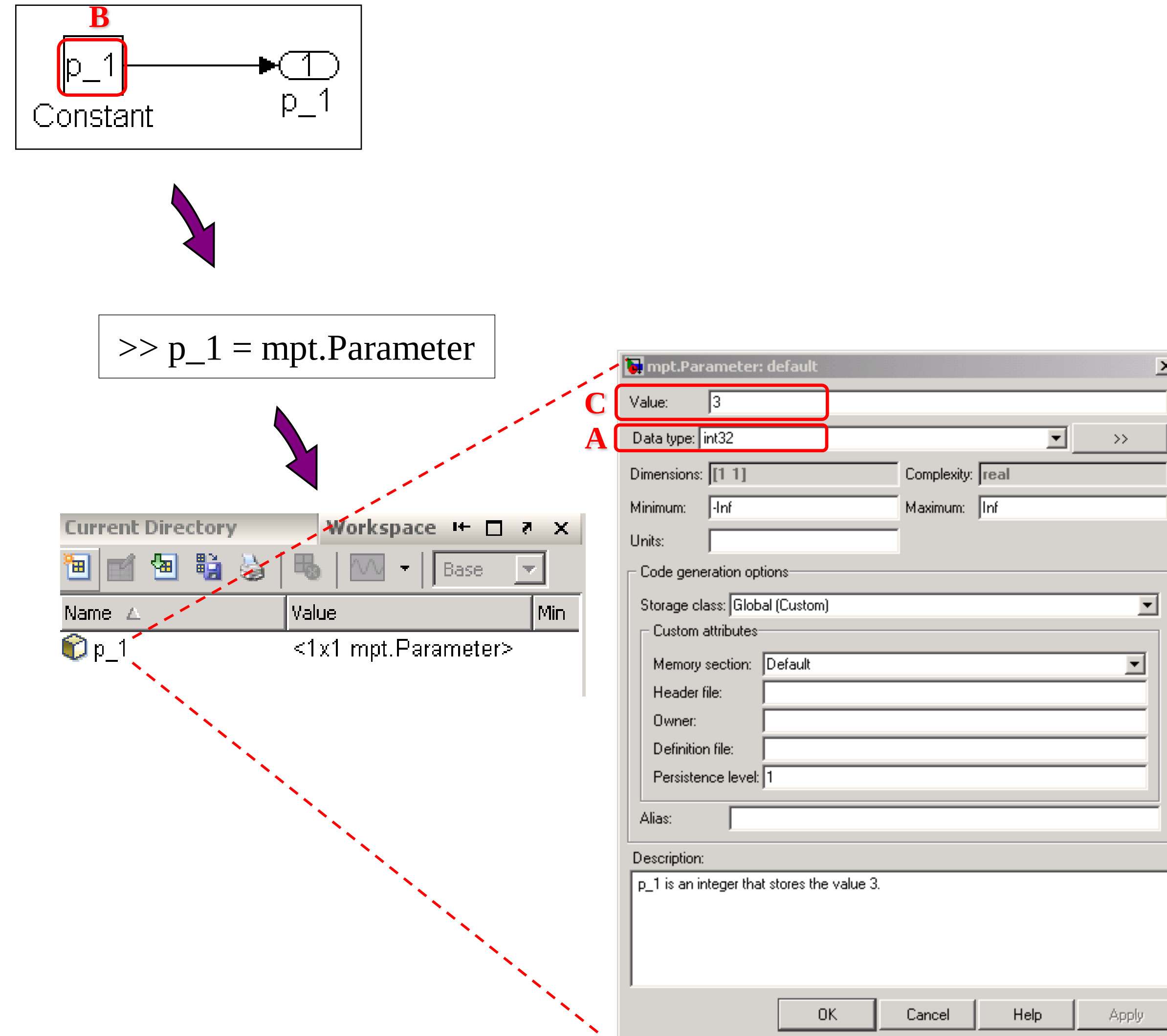
Data Declaration

Creation : Parameters

C- Code Pattern, Parameter

A B C

int p_1 = 3;



Generated Code, Parameter

A B C

int32_T p_1 = 3;

rtwdemo_intparam_sdo.mdl

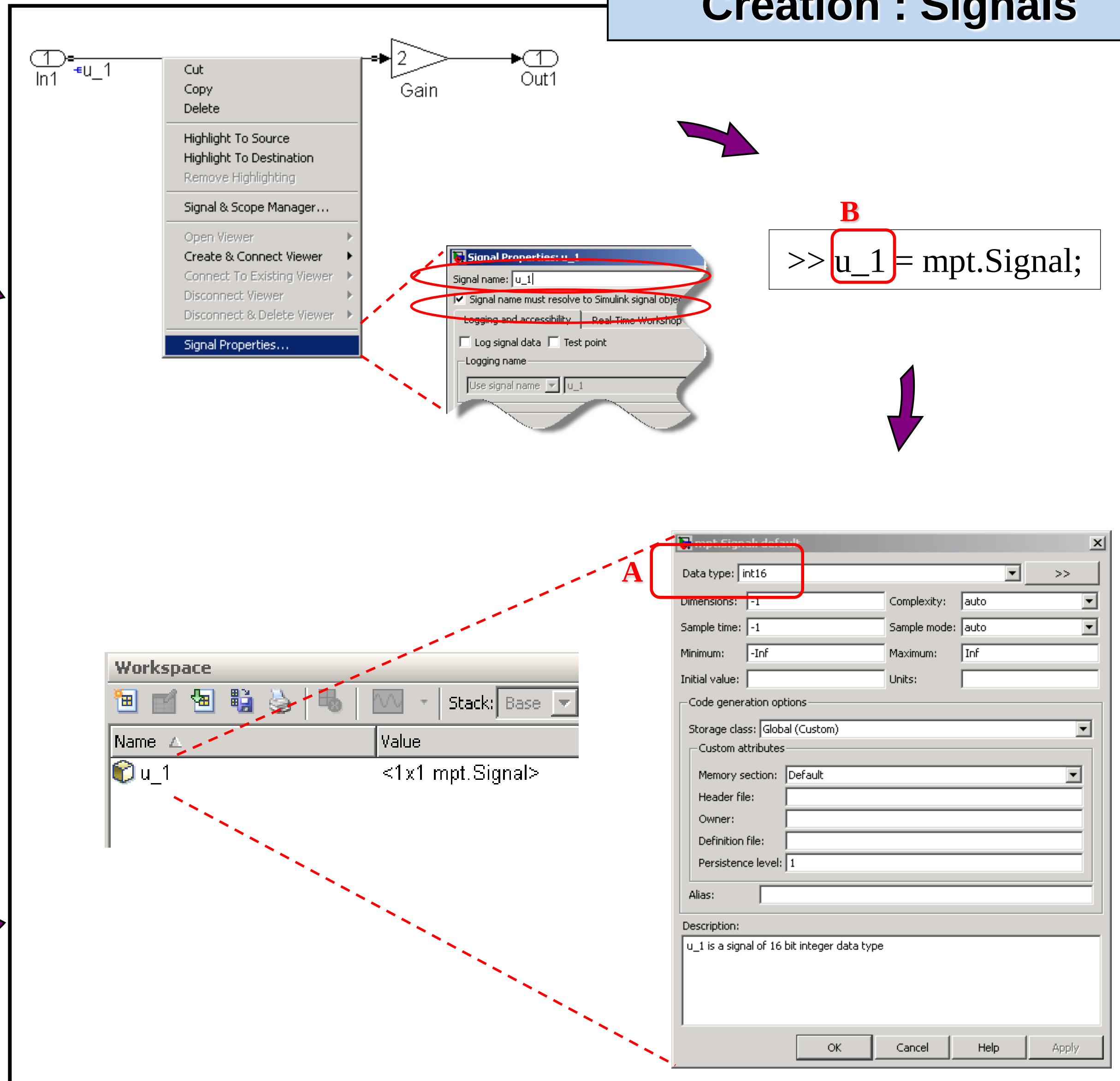
Data Declaration

Creation : Signals

C- Code Pattern, Signal

A B

int u_1;



Generated Code, Signal

A B

int16_T u_1;

rtwdemo_intsignal.mdl

C Code Pattern: Data-Type Conversion

Code Pattern	**Modeling Methodology**
y_1 = (double) u_1 ; Description: *u_1 is a 32 bit signed integer (int32). The typecasting operation converts an int32 to a double.*	1. Use Simulink Data-Type Conversion Block.
	2. Usage within Stateflow Chart
	3. Usage within Embedded MATLAB (EML) Block
	4. Other type conversion methods

Return to Table of Contents

Type-Conversion

Creation

SL | SF | EML

C-Code Pattern

y_1 = (double) u_1 ;

Generated Code

y_1 = (real_T)u_1;

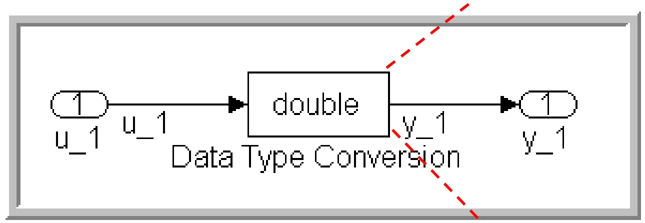
NOTE : real_T is a Real-Time Workshop Embedded Coder Typedef for double.

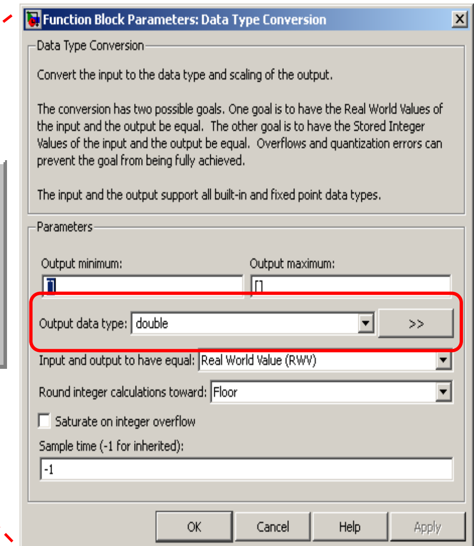
rtwdemo_typeconvsl.mdl

rtwdemo_typeconvsf.mdl

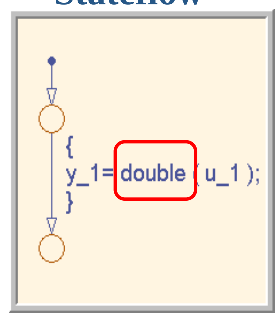
rtwdemo_typeconveml.mdl

Simulink





Stateflow



Embedded MATLAB Function

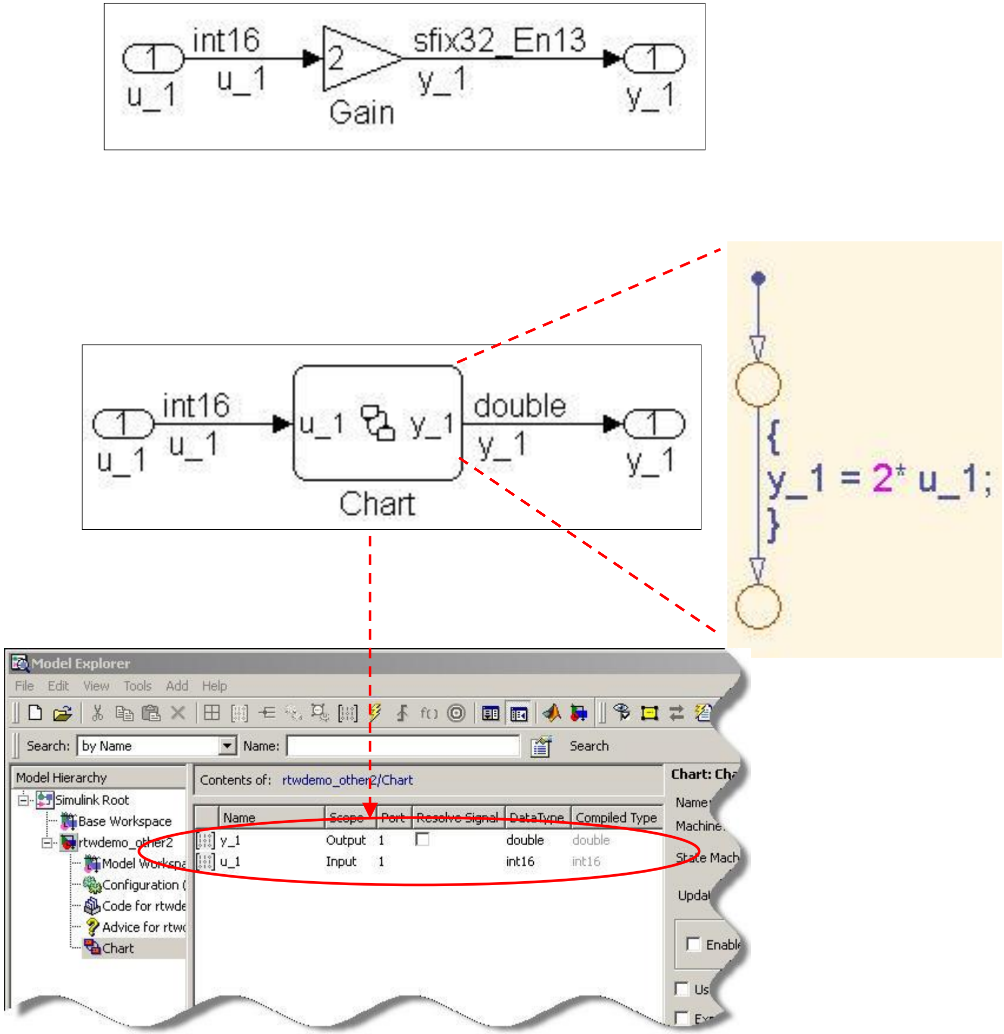
```
function y_1 = typeconv(u_1)

y_1 = double( u_1 );
```

Type-Conversion

Other type-conversion mechanisms

Example 1:

Example 2:



rtwdemo_other.mdl

rtwdemo_other2.mdl

C Code Pattern: Type Qualifiers

Code Pattern, Parameter	Modeling Methodology
const double p_1 = 9.8 ; ***Description:*** *The aim is to obtain declarations such as the one above in the generated code. p_1 is a constant parameter that has a double data-type.*	1. Create a parameter in the MATLAB workspace and make it tunable.
	2. Create a data Object and use the Const custom storage class.

Return to Table of Contents

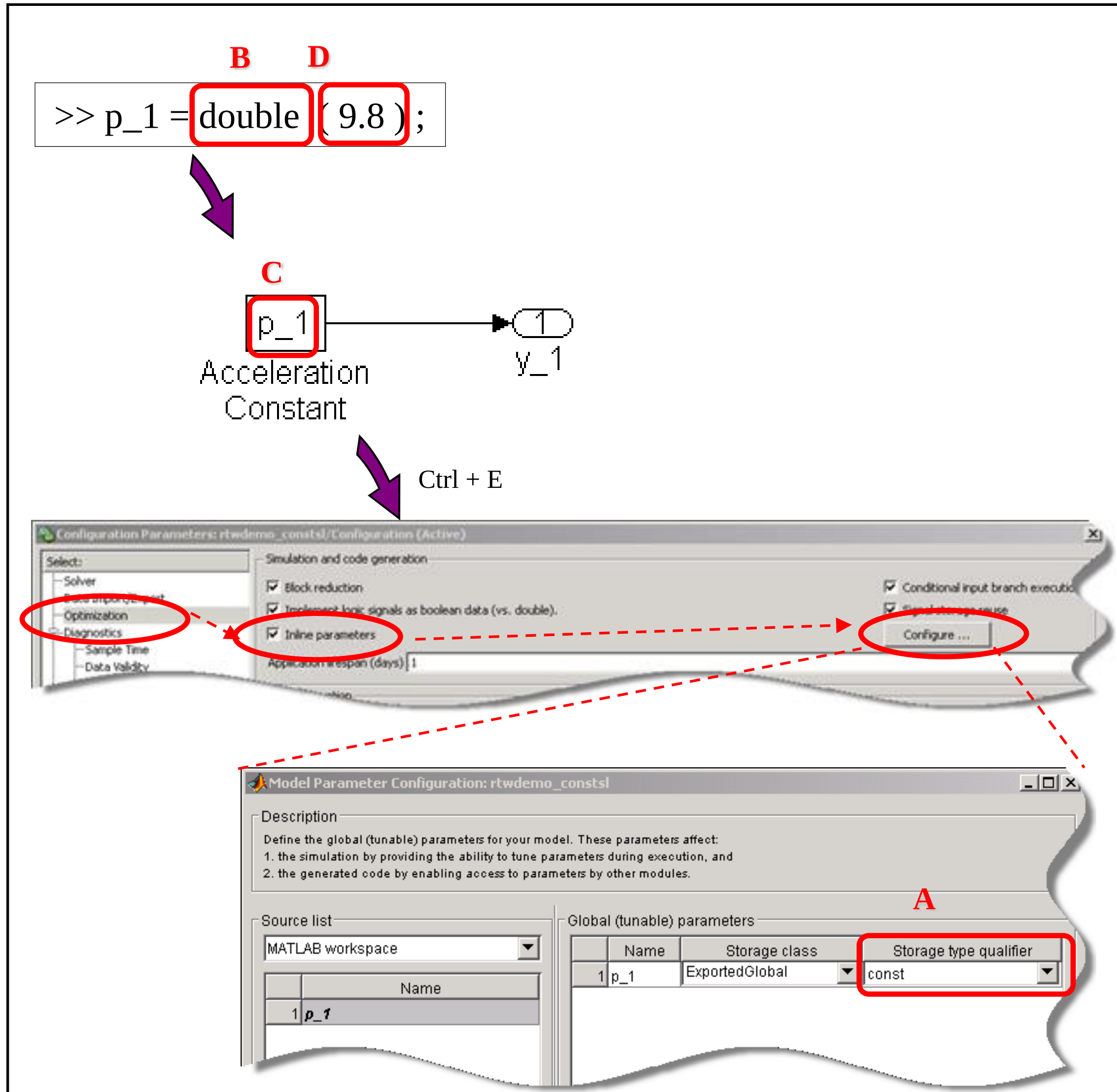
Type Qualifiers

Tunable Parameters

C- Code Pattern, Parameter

A B C D

const double p_1 = 9.8;



Generated Code, Parameter

A B C D

const real_T p_1 = 9.8 ;

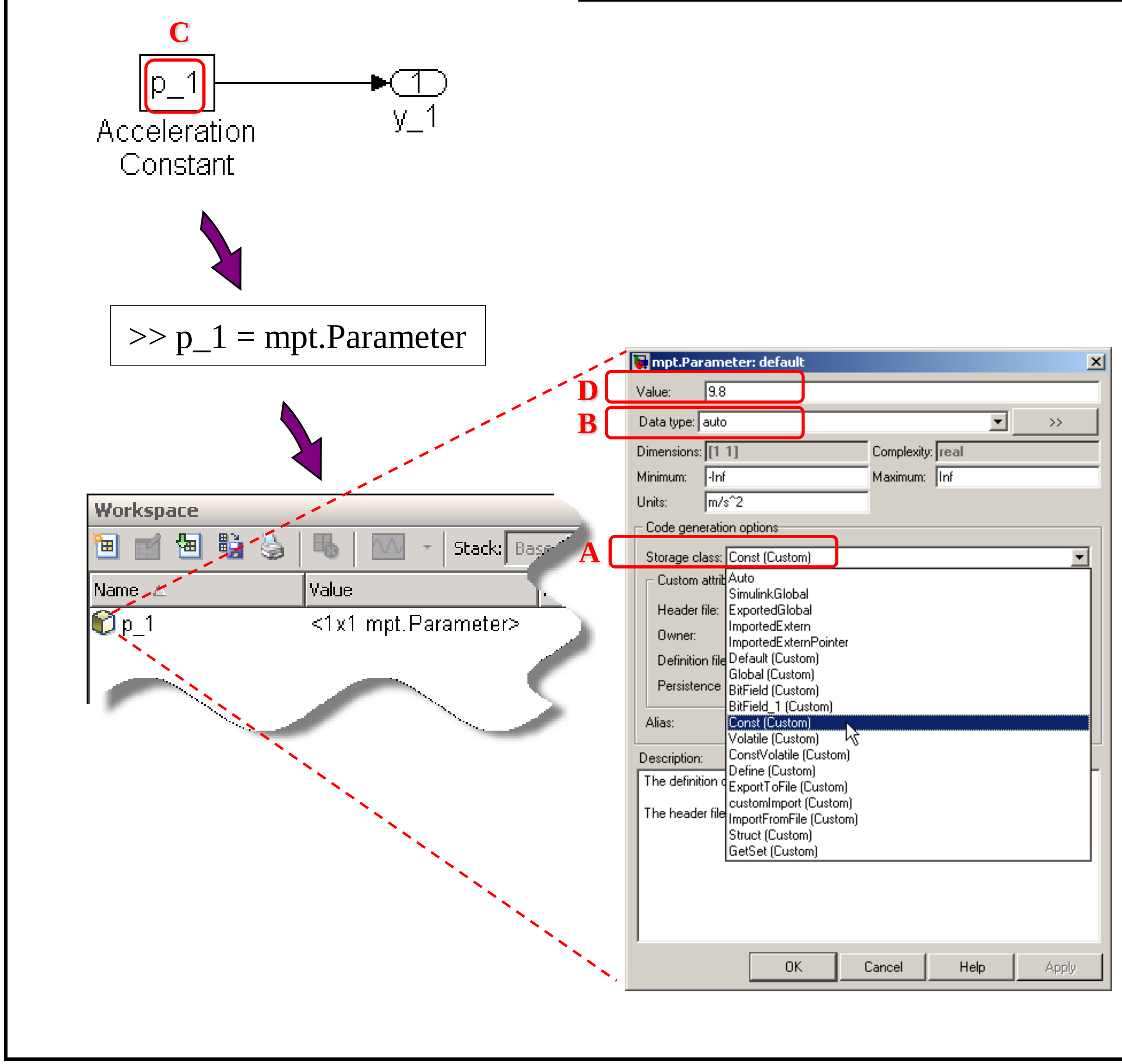
rtwdemo_constsl.mdl

Type Qualifiers

Simulink Data Object & CSC

C- Code Pattern, Parameter

A	B	C		D	
const	double	p_1	=	9.8	;



Generated Code, Parameter

A	B	C		D	
const	real_T	p_1	=	9.8	;

rtwdemo_constsl_sdo.mdl

C Code Pattern: Relational & logical Operators

Code Pattern, Relational Operator	Modeling Methodology
y_1 = (u_1 > u_2) ;	1. Use Simulink Relational & Logical Operator Blocks.
	2. Usage within Stateflow Chart (primarily for transitions)
Description: *In this example a relational-operation is shown.*	3. Usage within Embedded MATLAB (EML) Block

Code Pattern, Logical Operator	Modeling Methodology
	1. Use Simulink Relational & Logical Operator Blocks.
y_1 = (u_1 || u_2);	2. Usage within Stateflow Chart (primarily for transitions)
Description: *u_1, u_2 are Booleans. In this example a logical-operation is shown.*	3. Usage within Embedded MATLAB (EML) Block

Return to Table of Contents

Relational & Logical Operations

Creation

SL	SF	EML

C- Code Pattern, Relational Operator

A

y_1 = (u_1 $\boxed{>}$ u_2) ;

C- Code Pattern, Logical Operator

B

y_1 = (u_1 $\boxed{||}$ u_2) ;

Generated Code, Relational Operator

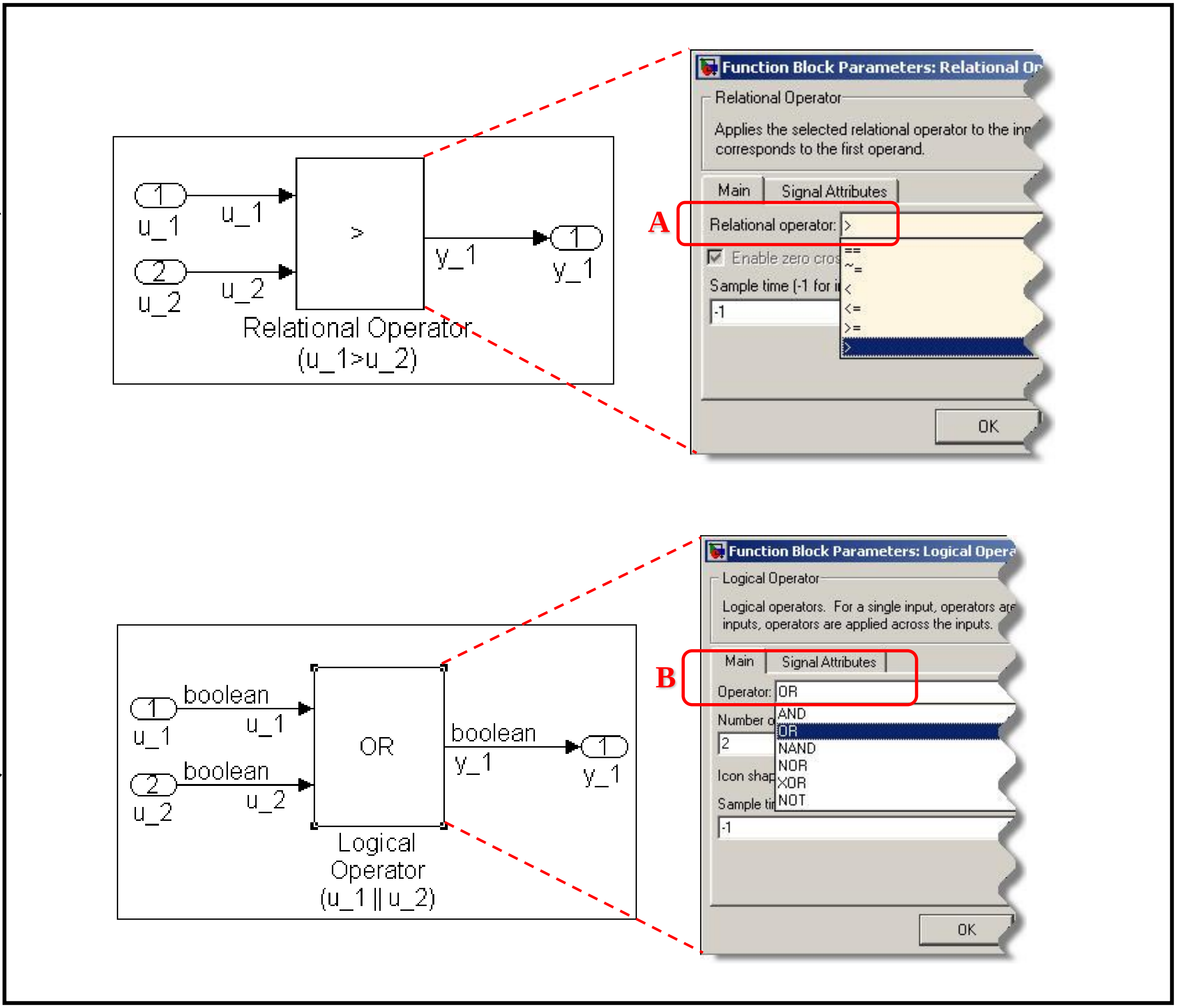
A

y_1 = (u_1 $\boxed{>}$ u_2);

Generated Code, Logical Operator

B

y_1 = (u_1 $\boxed{||}$ u_2);



rtwdemo_relationalsl.mdl
rtwdemo_logicalSL.mdl

Relational & Logical Operations

Creation

SL	SF	EML

C- Code Pattern, Relational Operator

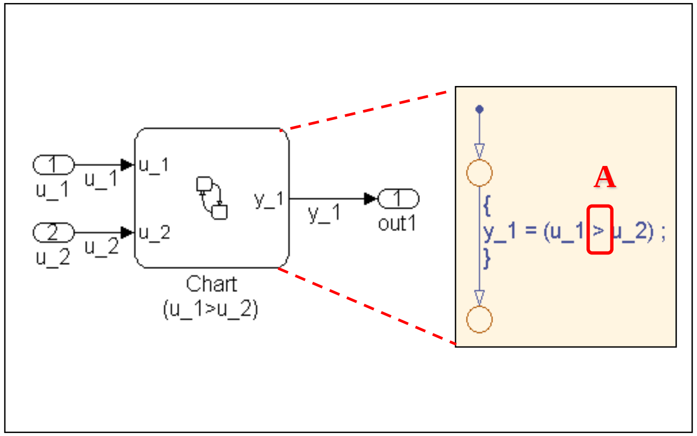
A

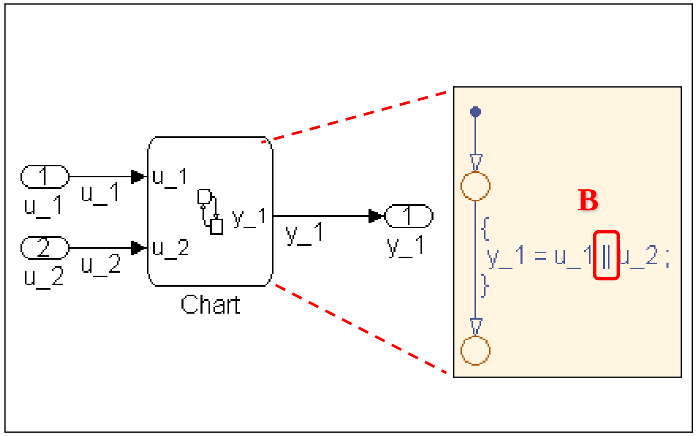
y_1 = (u_1 > u_2) ;

C- Code Pattern, Logical Operator

B

y_1 = (u_1 || u_2);





Generated Code, Relational Operator

A

y_1 = (u_1 > u_2);

Generated Code, Logical Operator

B

y_1 = (u_1 || u_2);

rtwdemo_relationalsf.mdl

rtwdemo_logicalORsf.mdl

Relational & Logical Operations

Creation

SL	SF	EML

C- Code Pattern, Relational Operator

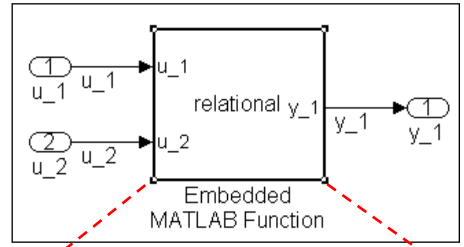
A

y_1 = (u_1 > u_2) ;

C- Code Pattern, Logical Operator

B

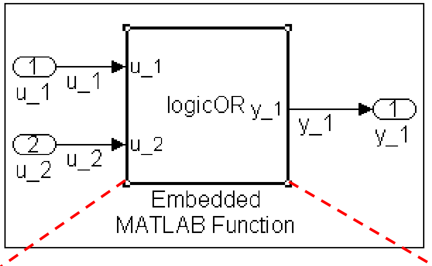
y_2 = (u_1 || u_2) ;




```
function y_1 = relational(u_1,u_2)
```

A

```
y_1 = (u_1 > u_2) ;
```




```
function y_1 = logicOR (u_1,u_2)
```

B

```
y_1 = u_1 || u_2 ;
```

Generated Code, Relational Operator

A

y_1 = u_1 > u_2;

Generated Code, Logical Operator

B

```
if ((!u_1) && (!u_2)) {
  y_1 = 0U;
} else {
  y_1 = 1U;
}
```

rtwdemo_relationaleml.mdl
rtwdemo_logicalOReml.mdl

C Code Pattern: Bitwise-Logic operations

Code Pattern, AND operation	Modeling Methodology
y_1 = u_1 & 0xD9 ;	1. Use Simulink Bitwise-Operator Block.
	2. Use Stateflow Chart
Description: u_1 is an 8 bit unsigned integer. In this example, a Bitwise-logical operation with a bit-mask is shown	3. Use Embedded MATLAB (EML) Block

Code Pattern, OR operation	Modeling Methodology
y_2 = u_2 | u_3;	1. Use Simulink Bitwise-Operator Block.
	2. Use Stateflow Chart
Description: u_2, u_3 are 8 bit unsigned integers. In this example a Bitwise-logical operation between two numbers is shown.	3. Use Embedded MATLAB (EML) Block

Return to Table of Contents

C- Code Pattern, AND

A B

y_1 = u_1 & 0xD9;

C- Code Pattern, OR

C

y_2 = u_2 | u_3;

Generated Code, AND

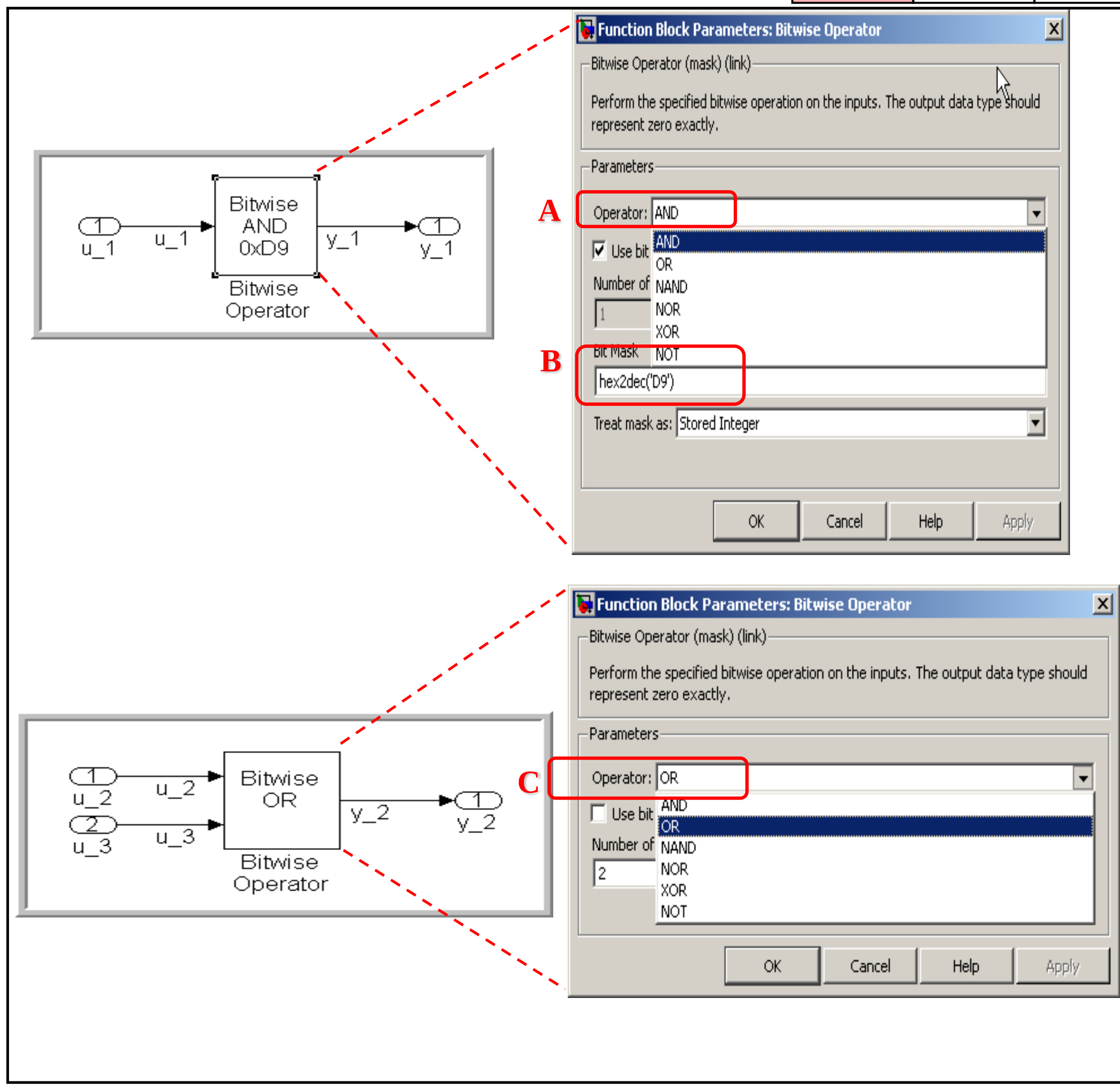
A B

y_1 = (uint8_T)(u_1 & 217U);

Generated Code, OR

C

y_2 = (uint8_T)(u_2 | u_3);



rtwdemo_logicANDsl.mdl

rtwdemo_logicORsl.mdl

C- Code Pattern, AND

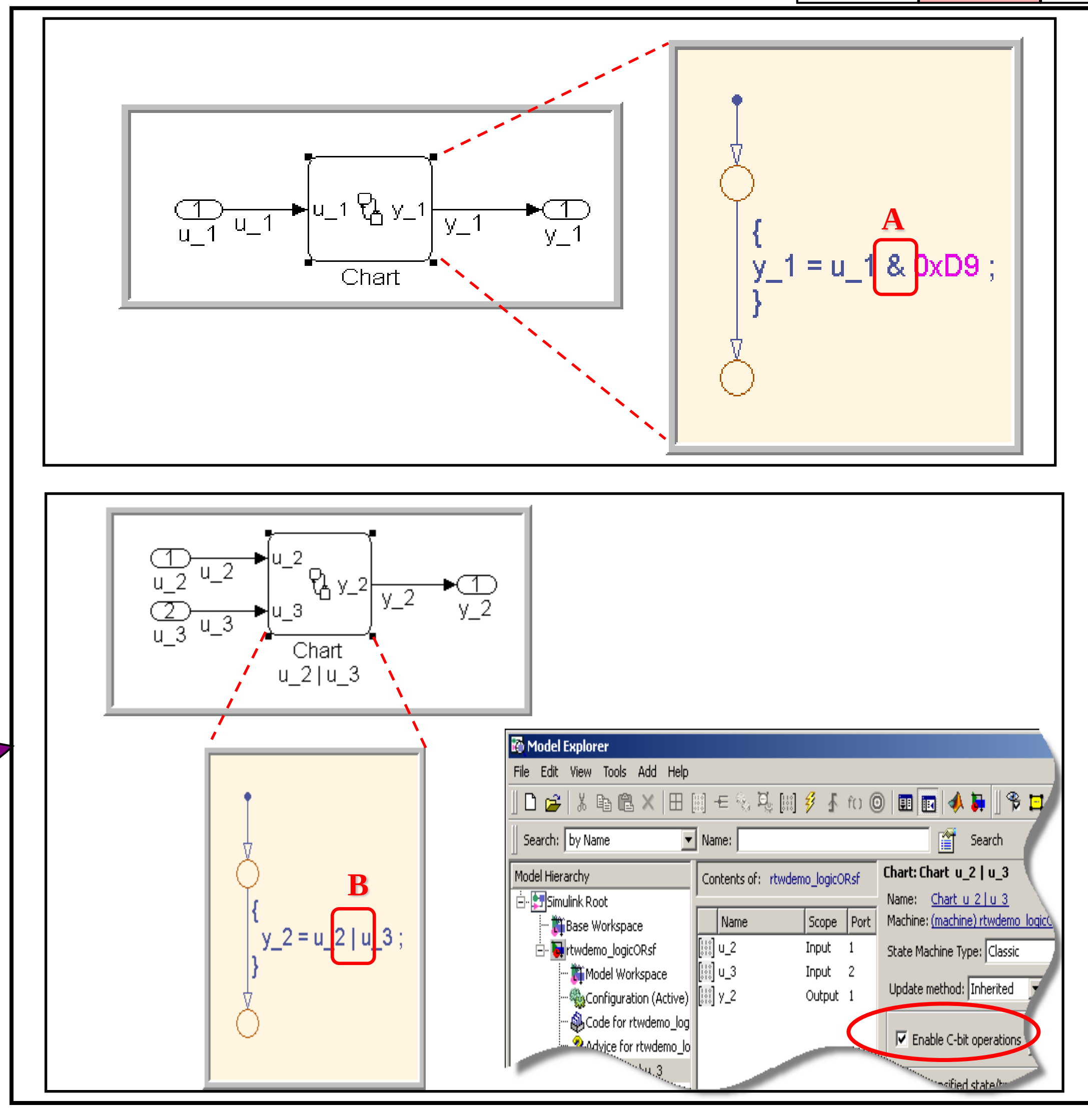
A

y_1 = u_1 & 0xD9 ;

C- Code Pattern, OR

B

y_2 = u_2 | u_3;



Generated Code, AND

A

y_1 = (uint8_T)(u_1 & 0xD9);

Generated Code, OR

B

y_2 = (uint8_T)(u_2 | u_3);

rtwdemo_logicANDsf.mdl
rtwdemo_logicORsf.mdl

Bitwise-Logic

Creation

SL	SF	EML

C- Code Pattern, AND

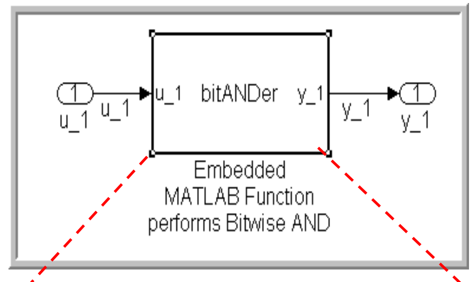
A

y_1 = u_1 & 0xD9 ;

C- Code Pattern, OR

B

y_2 = u_2 | u_3;




```
function y_1 = bitANDer (u_1)
```

A

```
y_1 = bitand (u_1,217) ;
```

Generated Code, AND

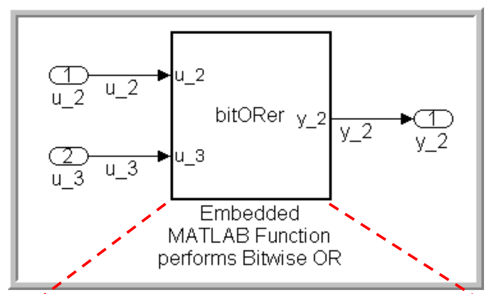
A

y_1 = (uint8_T)(u_1 & 217);

Generated Code, OR

B

y_2 = (uint8_T)(u_2 | u_3);




```
function y_2 = bitORer(u_2,u_3)
```

B

```
y_2 = bitor (u_2,u_3);
```

rtwdemo_logicANDeml.mdl

rtwdemo_logicOReml.mdl

C Code Pattern: If-Then-Else

Code Pattern	Modeling Methodology
if (u_1 > u_2) { y_1 = u_1; } else { y_1 = u_2; }	1. Use Simulink Switch Block 2. Use Stateflow Chart 3. Use Embedded MATLAB (EML) Block

Return to Table of Contents

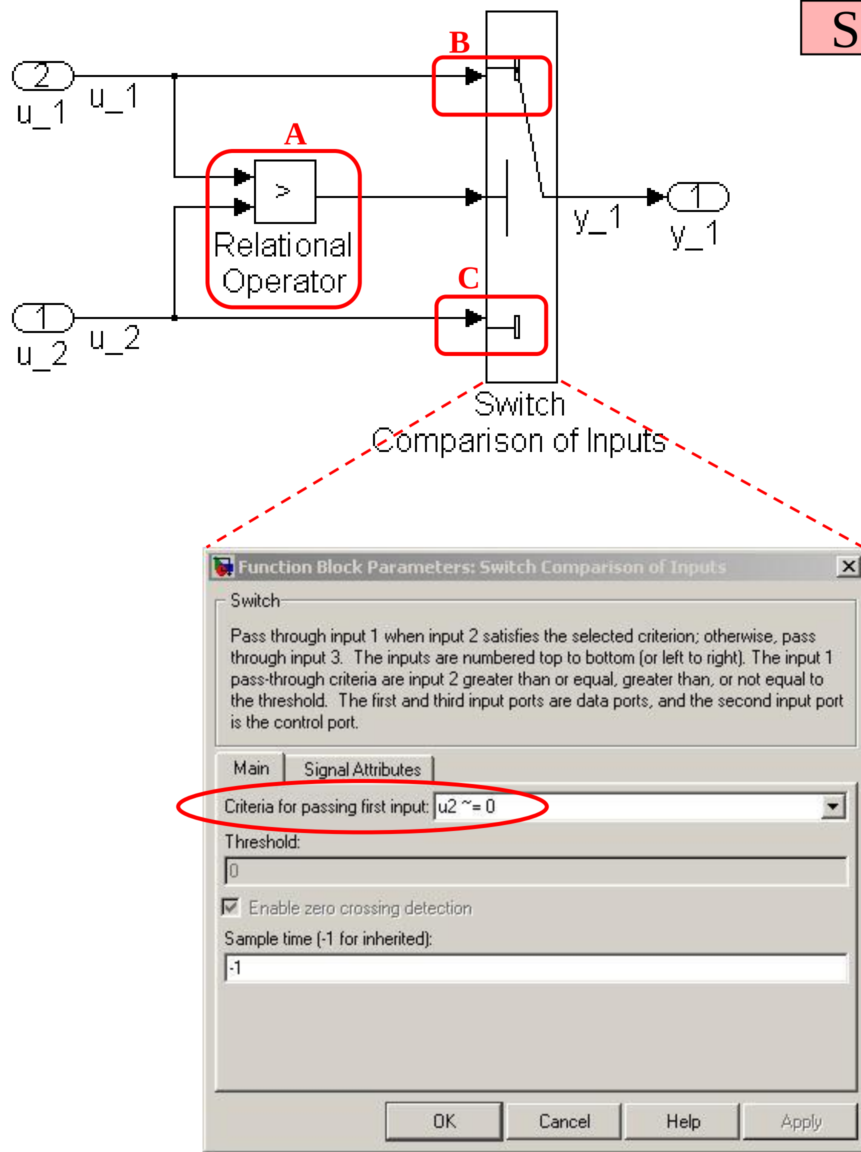
If-Then-Else

Creation

SL	SF	EML

C- Code Pattern

```
A  if ( u_1 > u_2 )
   {
B    y_1 = u_1;
   }
   else
   {
C    y_1 = u_2;
   }
```



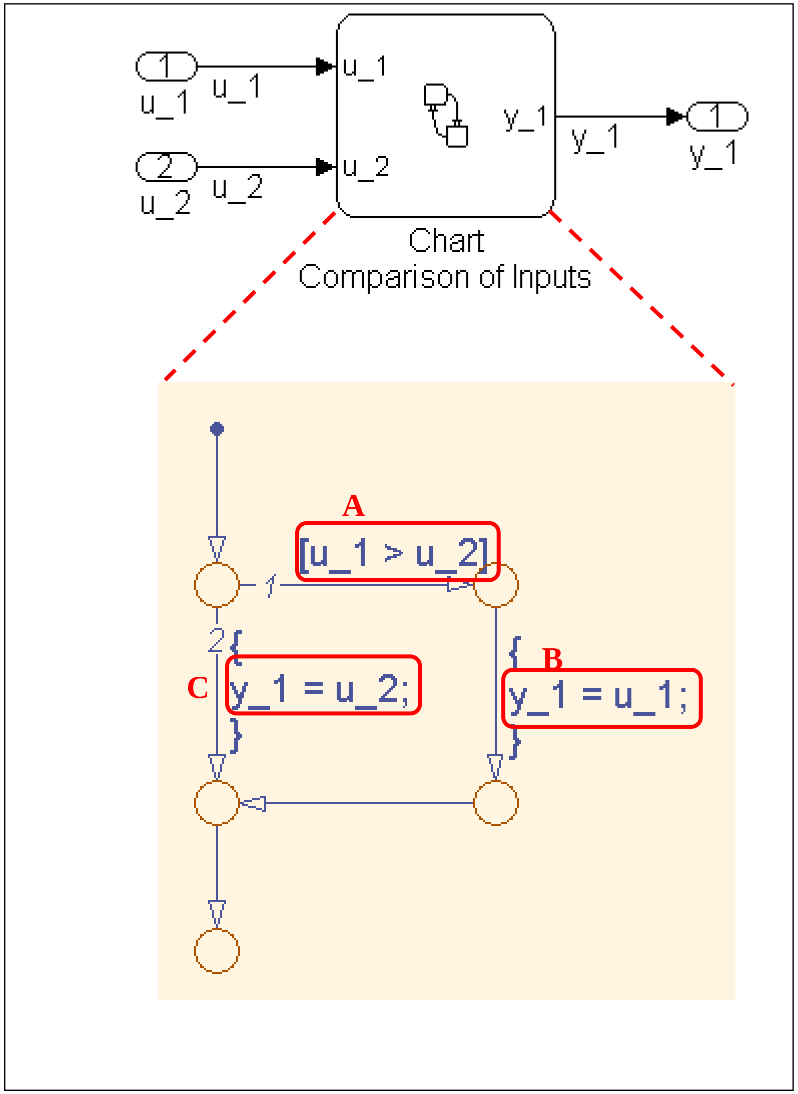
Generated ode

```
A  if (u_1 > u_2)
   {
B    y_1 = u_1;
   }
   else {
C    y_1 = u_2;
   }
```

rtwdemo_ifelse.mdl

C- Code Pattern

```
A  if (u_1 > u_2 )
    {
B     y_1 = u_1;
    }
    else
    {
C     y_1 = u_2;
    }
```



Generated Code

```
A   if (u_1 > u_2)
    {
B     y_1 = u_1;
    }
    else
    {
C     y_1 = u_2;
    }
```

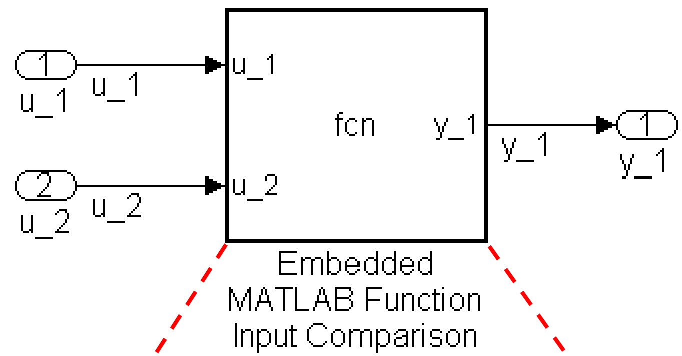
If-Then-Else

Creation

SL	SF	EML

C- Code Pattern

```
A  if ( u_1 > u_2 )
   {
B    y_1 = u_1;
   }
   else
   {
C    y_1 = u_2;
   }
```




```
function y_1 = fcn(u_1,u_2)
A
  if (u_1 > u_2)

B  y_1 = u_1 ;
  else
C  y_1 = u_2 ;
  end
```

Generated Code

```
A  if (u_1 > u_2)
   {
B    y_1 = u_1;
   }
   else {
C    y_1 = u_2;
   }
```

rtwdemo_ifelseeml.mdl

C Code Pattern: Switch-Case

Code Pattern	Modeling Methodology
switch (u_1) { case 2: y_1 = u_2; break; case 3: y_1 = u_3 ; break; default: y_1 = u_4; break; }	1. Use Simulink Switch-Case Block 2. Use Embedded MATLAB (EML) Block

Return to Table of Contents

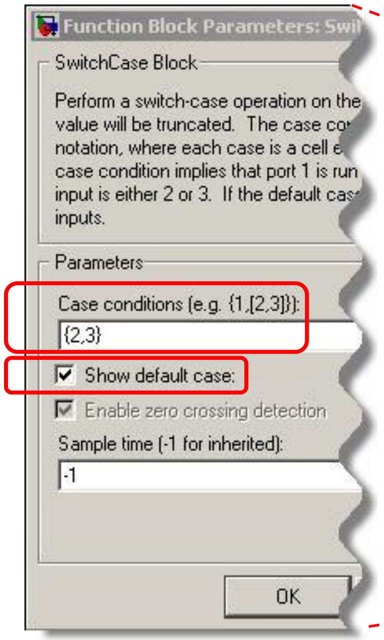
Switch-Case

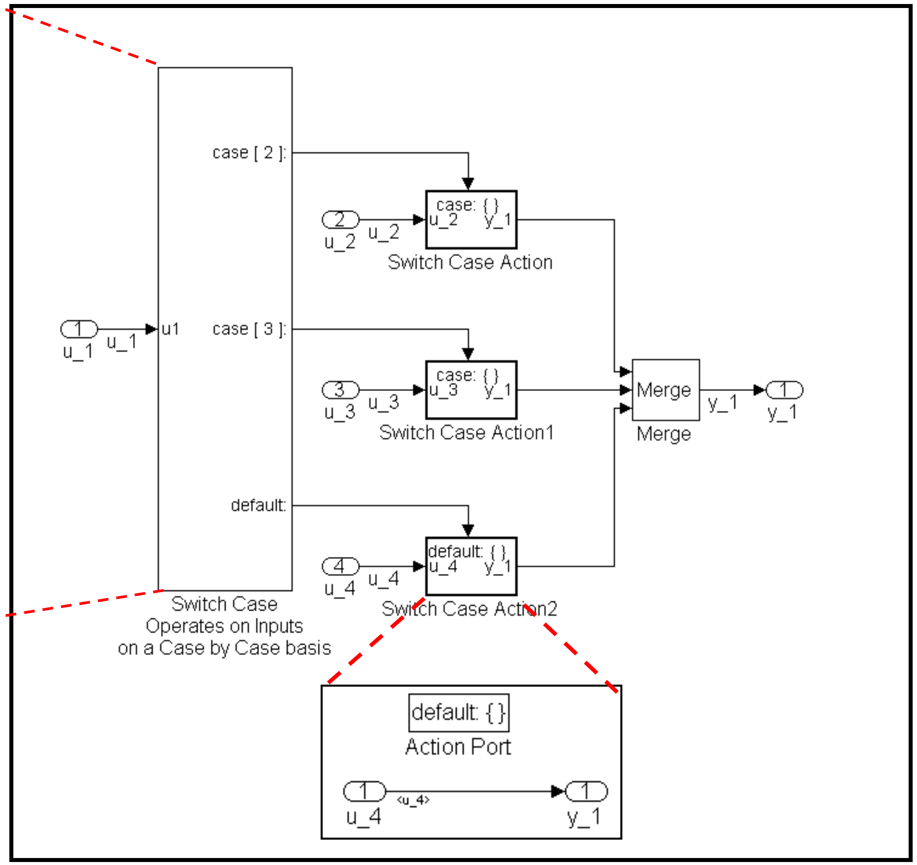
Creation

SL	EML

C- Code Pattern

```
switch (u_1)
{
A  case 2:
        y_1 = u_2;
        break;
   case 3:
        y_1 = u_3 ;
        break;
B  default:
        y_1 = u_4;
        break;
}
```





Generated Code

```
switch (u_1) {
A  case 2:
        y_1 = u_2;
        break;
   case 3:
        y_1 = u_3;
        break;
B  default:
        y_1 = u_4;
        break;
}
```

rtwdemo_switchsl.mdl

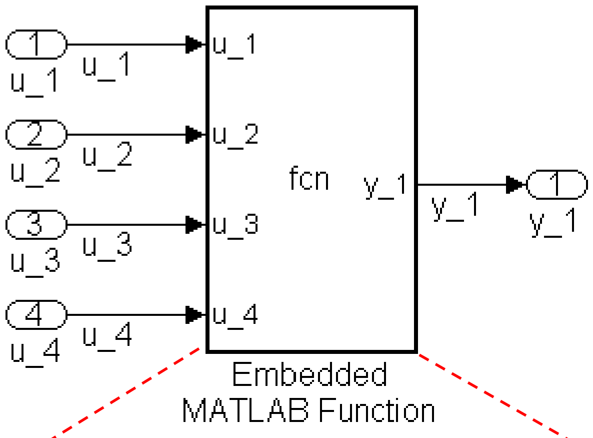
Switch-Case

Creation

SL	EML

C- Code Pattern

```
switch (u_1)
{
  case 2:
         y_1 = u_2;
         break;
  case 3:
         y_1 = u_3 ;
         break;
  default:
         y_1 = u_4;
         break;
}
```




```
function y_1 = fcn(u_1,u_2,u_3,u_4)

switch u_1
   case 2
      y_1= u_2;
   case 3
      y_1 = u_3;
   otherwise
      y_1 = u_4;
end
```

Generated Code

```
switch (u_1) {
  case 2U:
         y_1 = u_2;
         break;
  case 3U:
         y_1 = u_3;
         break;

  default:
         y_1 = u_4;
         break;
}
```

rtwdemo_switcheml.mdl

C Code Pattern: For - Loop

Code Pattern	Modeling Methodology
`y_1 = 0;` `for(inx = 0; inx < 10; inx++)` `{` `y_1 = u_1 [ inx ] + y_1 ;` `}` *Description: for-loop for summing up elements of an array. In this example, u_1 is a vector with dimension 10.*	1. Use Simulink For-Iterator Block
	2. Use Stateflow Chart
	3. Use Embedded MATLAB (EML) Block

Return to Table of Contents

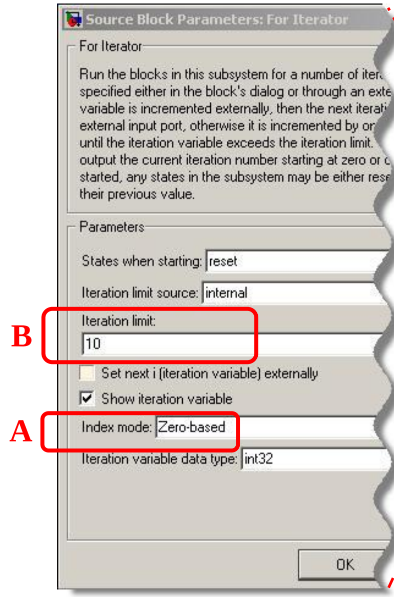
For-Loop

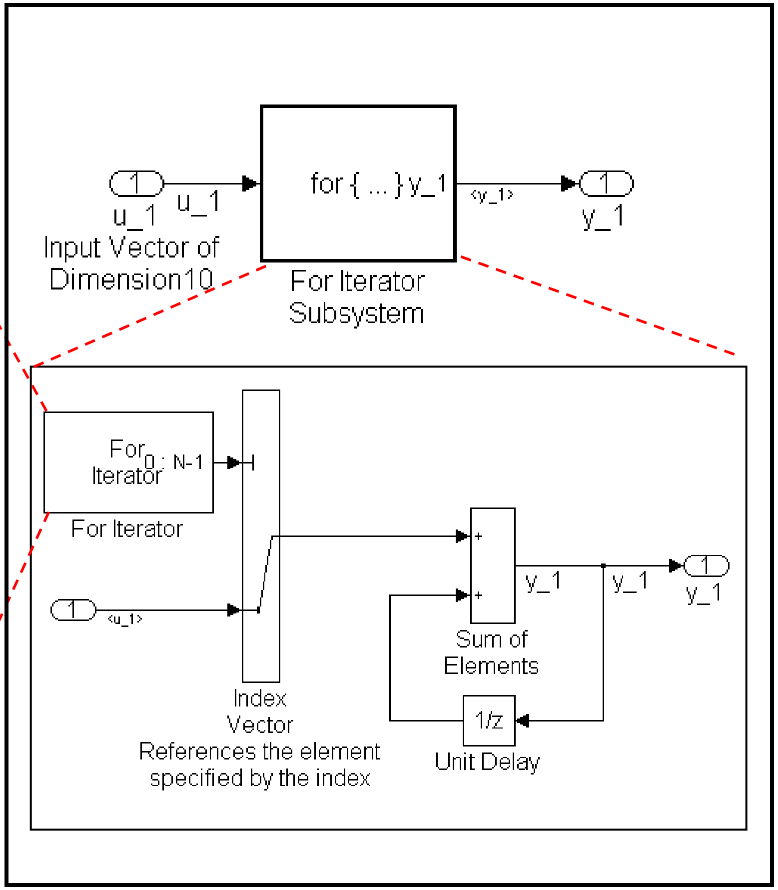
Creation

SL	SF	EML

C- Code Pattern

```
y_1 = 0;
     A          B
for(inx = 0; inx < 10; inx++)
{
        y_1 = u_1 [ inx ] + y_1 ;
}
```





Generated Code

```
                 A              B
x_1 = 0.0;
  for (s1_iter = 0; s1_iter < 10; s1_iter++) {
    y_1 = u_1[s1_iter] + x_1;
    x_1 = y_1;
  }
```

rtwdemo_forsl.mdl

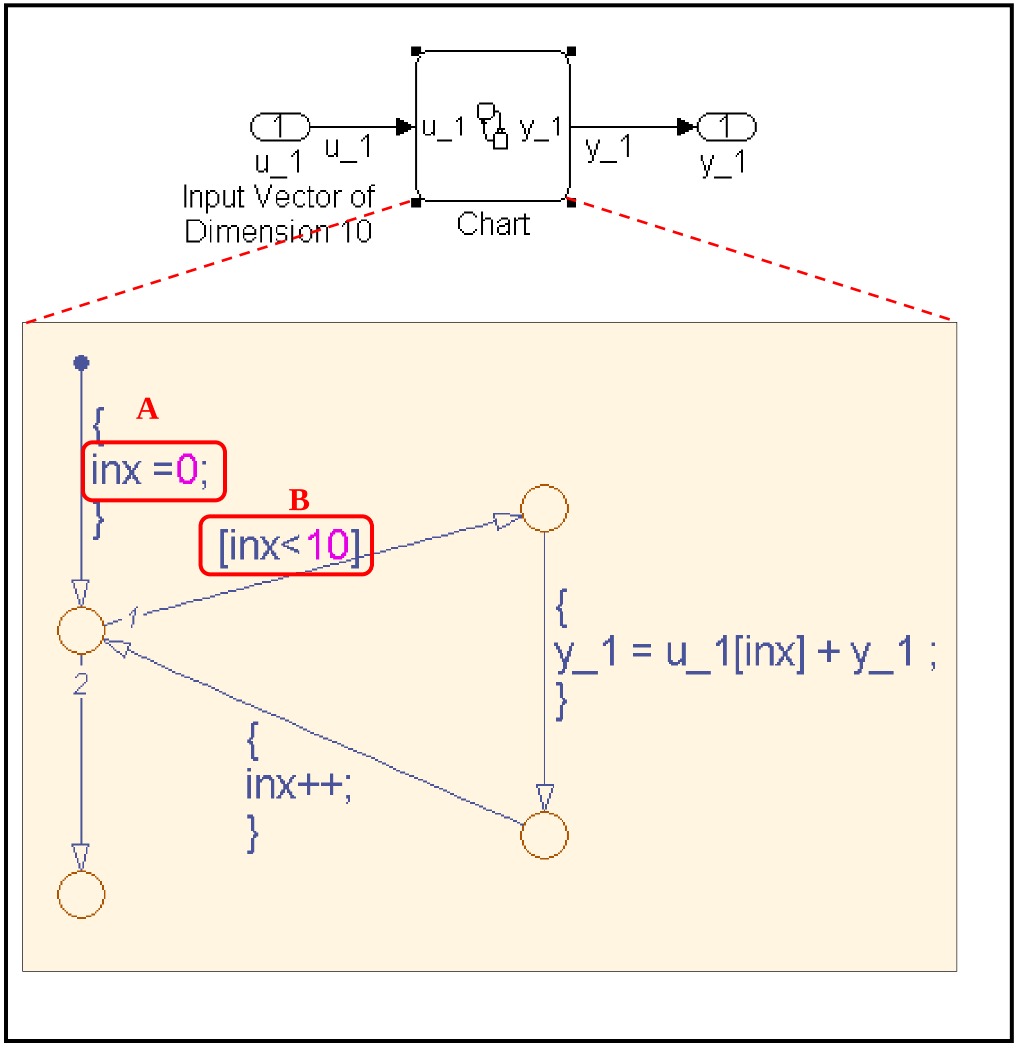
For-Loop

Creation

SL	SF	EML

C- Code Pattern

```
y_1 = 0;
     A           B
for(inx = 0; inx < 10; inx++)
{
        y_1 = u_1 [ inx ] + y_1 ;
}
```



Generated Code

```
          A              B
for (sf_inx = 0; sf_inx < 10; sf_inx++)
{
    y_1 = u_1[sf_inx] + y_1;
 }
```

rtwdemo_forloopsf.mdl

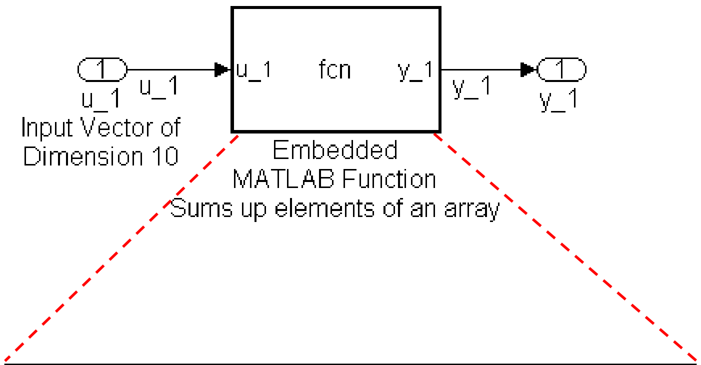
For-Loop

Creation

SL	SF	EML

C- Code Pattern

```
y_1 = 0;
     A          B
for(inx = 0; inx < 10; inx++)
{
        y_1 = u_1 [ inx ] + y_1 ;
}
```




```
function y_1 = fcn(u_1)

y_1 = 0;

for inx=1:10
    y_1 = u_1(inx) + y_1 ;
end
```

Generated Code

```
y_1 = 0.0;   A             B
for (eml_inx = 0; eml_inx < 10; eml_inx++)
{
     y_1 = u_1[eml_inx] + y_1;
}
```

rtwdemo_forloopeml.mdl

NOTE: In M-Script, the index is One-based

C Code Pattern: While - Loop

Code Pattern	Modeling Methodology
`while( flag && (num_iter<=100) ) { flag = func ( ); num_iter ++ ; }` *Description: A while loop used to ensure that a function returns "TRUE". The Number of iterations allowed is fixed and set to 100*	1. Use Simulink While Block
	2. Use Stateflow Chart
	3. Use Embedded MATLAB (EML) Block

Return to Table of Contents

While-Loop

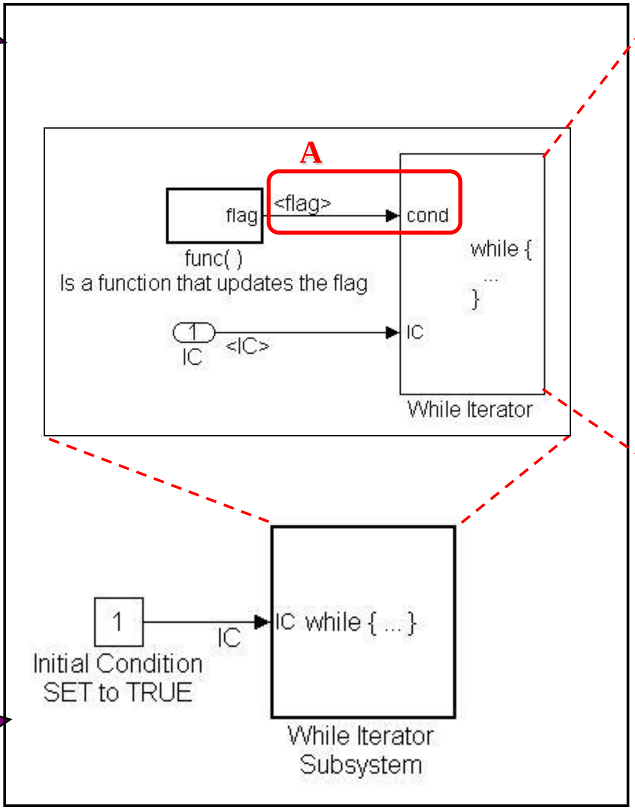
Creation

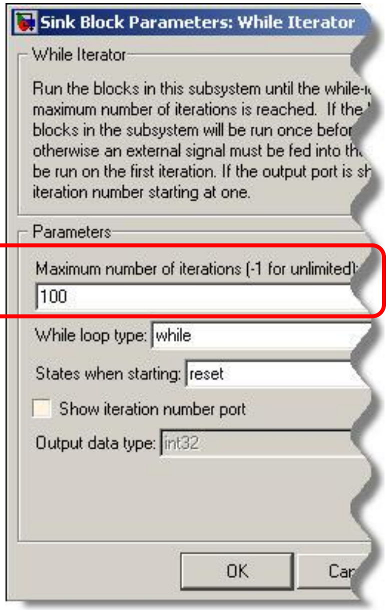
SL	SF	EML

C- Code Pattern

A B

```
while ( flag && (num_iter<= 100) )
{
   flag  = func ( );
   num_iter ++;
}
```





Generated Code

A B

```
while (loopCond && (s1_iter <= 100))
 {
     func();
     loopCond = flag;
     s1_iter++;
 }
```

NOTE: ‘flag’ is a global variable updated by the function func ( )

rtwdemo_whilesl.mdl

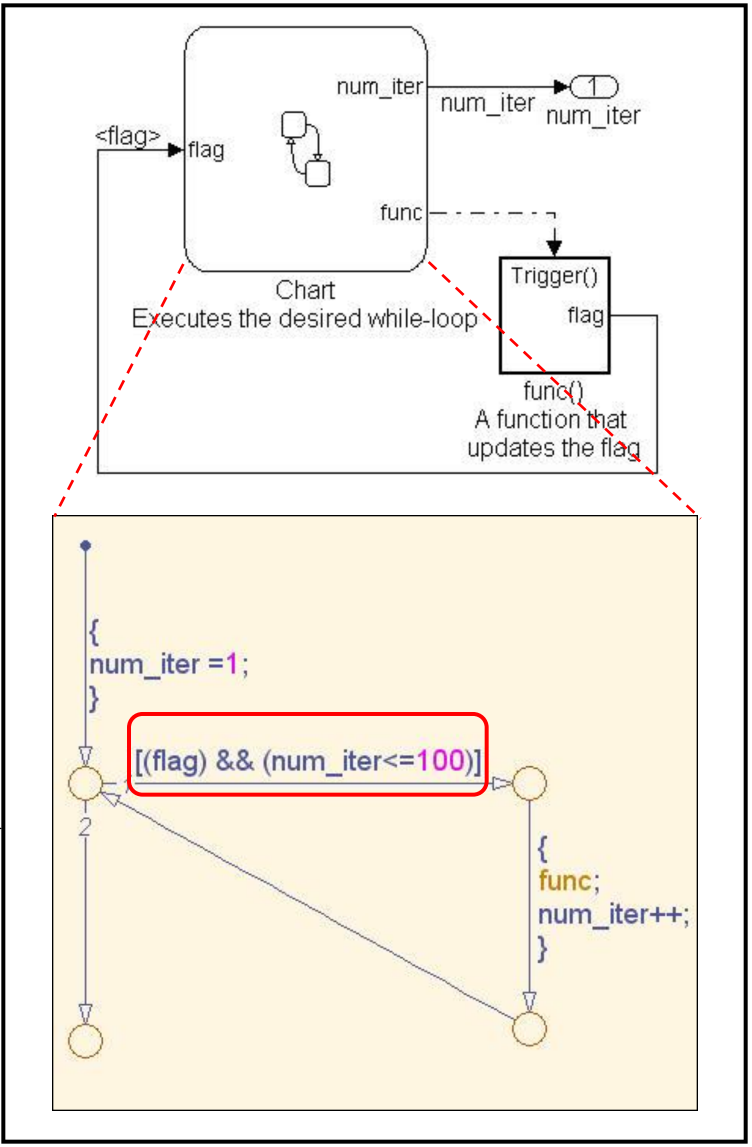
While-Loop

Creation

SL | SF | EML

C- Code Pattern

```
while ( flag && (num_iter<= 100) )
{
   flag  = func ( );
   num_iter ++;
}
```



Generated Code

```
while (flag && (num_iter <= 100))
{
  func();
  num_iter = num_iter + 1;
}
```

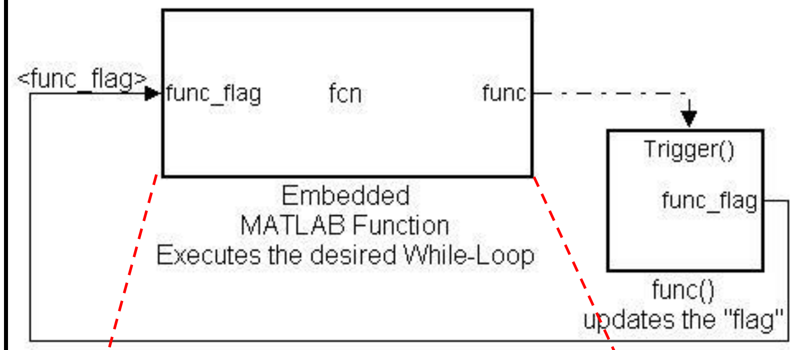
NOTE: ‘flag’ is a global variable updated by the function func ( )

rtwdemo_whilesf.mdl

While-Loop

C- Code Pattern

```
while ( flag && (num_iter<= 100) )
{
   flag  = func ( );
   num_iter ++;
}
```




```
function fcn(func_flag)

flag = true; num_iter = 1;

while flag && (num_iter<=100))
   func;
   flag = func_flag;
   num_iter = num_iter + 1;
end
```

Generated Code

```
for (eml_num_iter = 1; eml_flag && (eml_num_iter <=100); eml_num_iter++)
 {
    func();
    eml_flag = eml_func_flag;
 }
```

NOTE: The function func ( ) updates the flag

rtwdemo_whileeml.mdl

C Code Pattern: do-while Loop

Code Pattern	Modeling Methodology
`num_iter = 1;` `do {` `    flag = func ( );` `    num_iter++ ;` `  }` `while( flag && (num_iter<=100)) ;` *Description: A do-while loop used to ensure that a function returns "TRUE". The Number of iterations allowed is fixed and set to 100.*	1. Use Simulink While Block 2. Use Stateflow Chart

Return to Table of Contents

do while loop

Creation

SL | SF

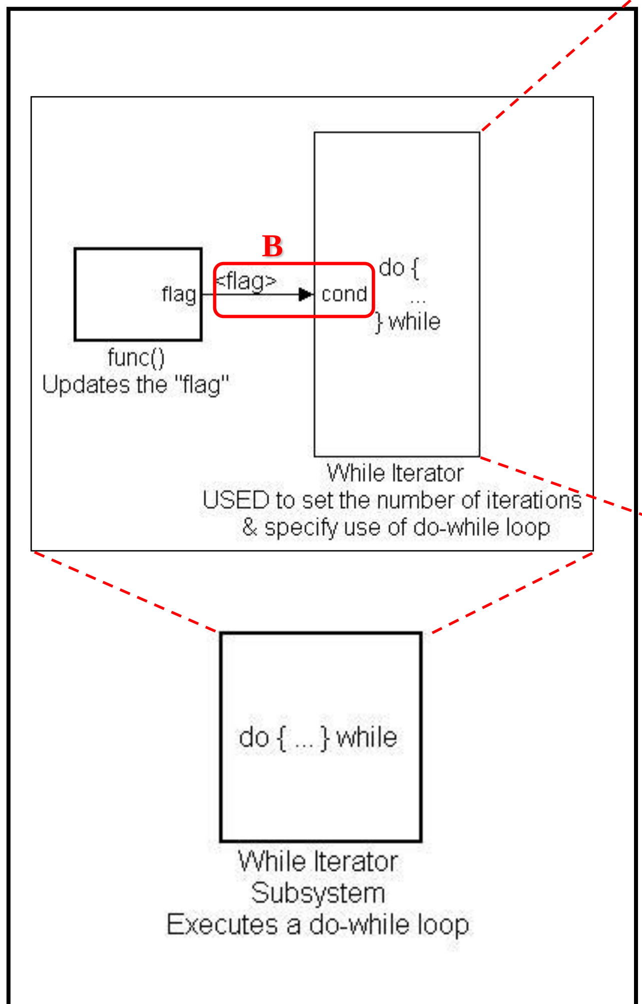
C- Code Pattern

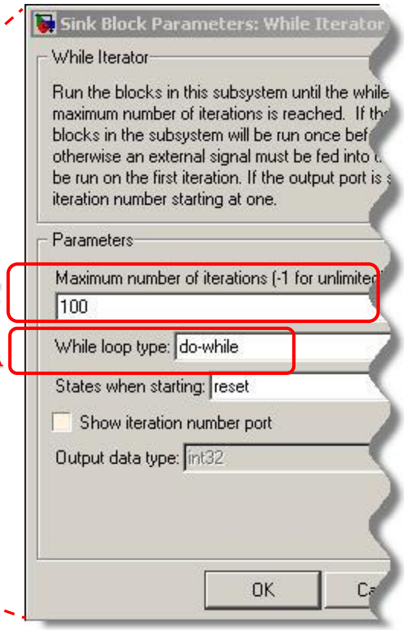
```
   num_iter = 1;
A  do {
        flag = func( ) ;
        num_iter++ ;
     }  B           C
   while (flag && (num_iter<=100));
```

Generated Code

```
   s1_iter = 1;
A  do {
        func();
        s1_iter++;
     }  B           C
   while (flag && (s1_iter <= 100));
```

NOTE: ‘flag’ is a global variable updated by the function func ( )





rtwdemo_dowhilesl.mdl

do while loop

Creation

SL	SF

C- Code Pattern

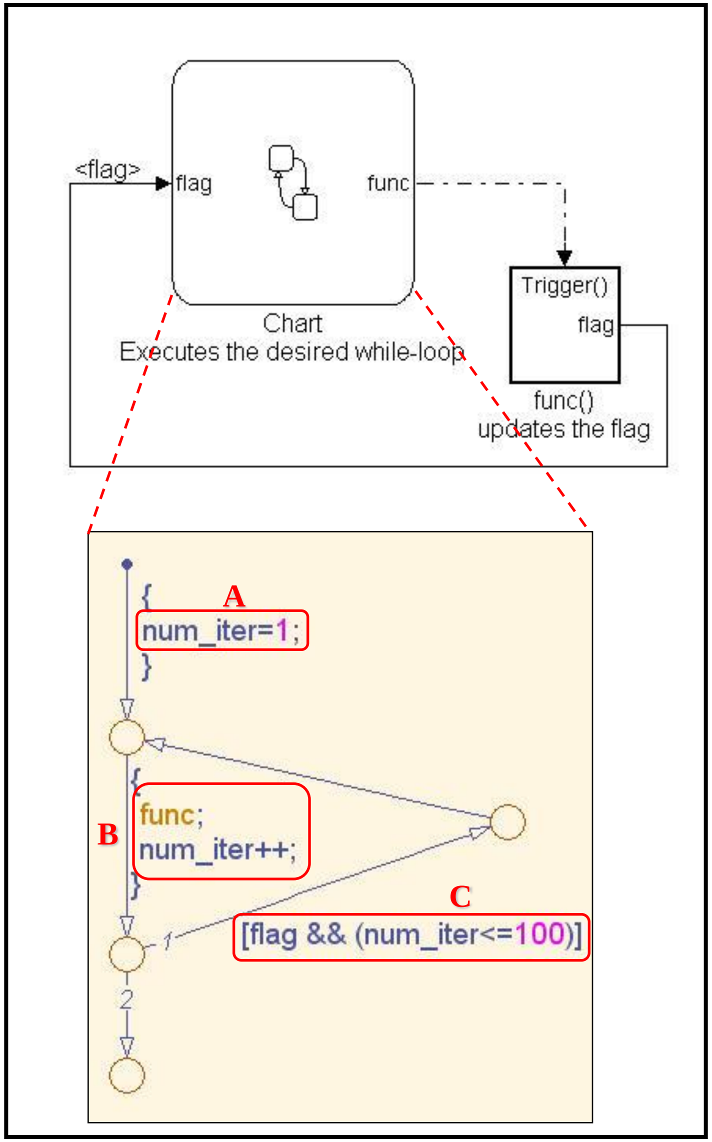
```
num_iter = 1;
do {
      flag = func( ) ;
      num_iter++ ;
   }
while (flag && (num_iter<=100)) ;
```

Generated Code

```
sf_num_iter = 1;
do {
   func();
   sf_num_iter++;
} while (flag && (sf_num_iter <= 100));
```

NOTE: 'flag' is a global variable updated by the function func ( )

rtwdemo_dowhilesf.mdl



C Code Pattern: Functions

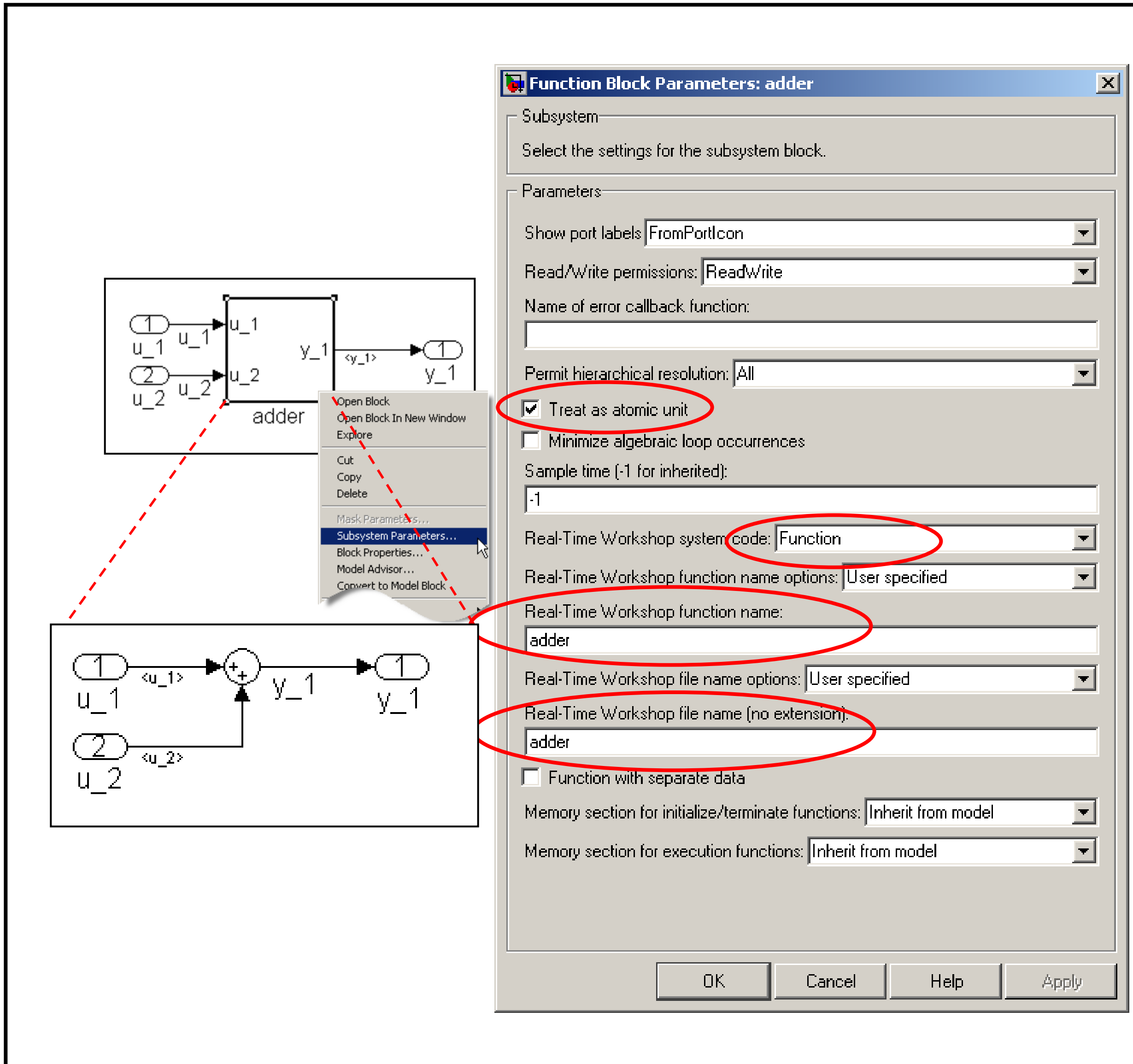
Code Pattern, void <fcn> (void)	Modeling Methodology
void adder (void) { y_1 = u_1 + u_2; } *Description: Obtain functions in the generated code that operate on global variables. Here u_1, u_2 and y_1 are all global variables.*	Create a subsystem and treat as atomic unit.

functions

Creation with Atomic Subsystem

C- Code Pattern

```
void adder (void)
{
    y_1 = u_1 + u_2;
}
```



Generated Code

<function>.c

```
void adder(void)
{
y_1 = u_1 + u_2;
}
```

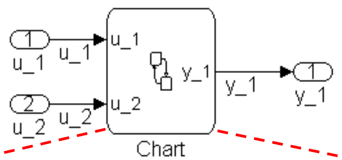
rtwdemo_voidvoidfcn.mdl

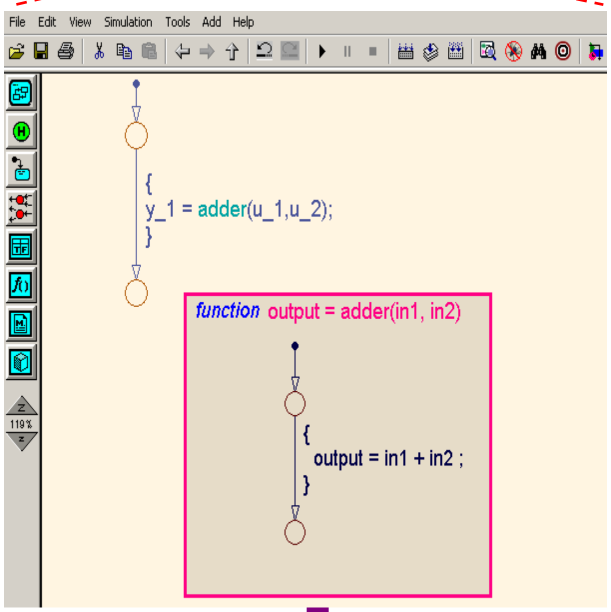
C Code Pattern: Function Prototyping

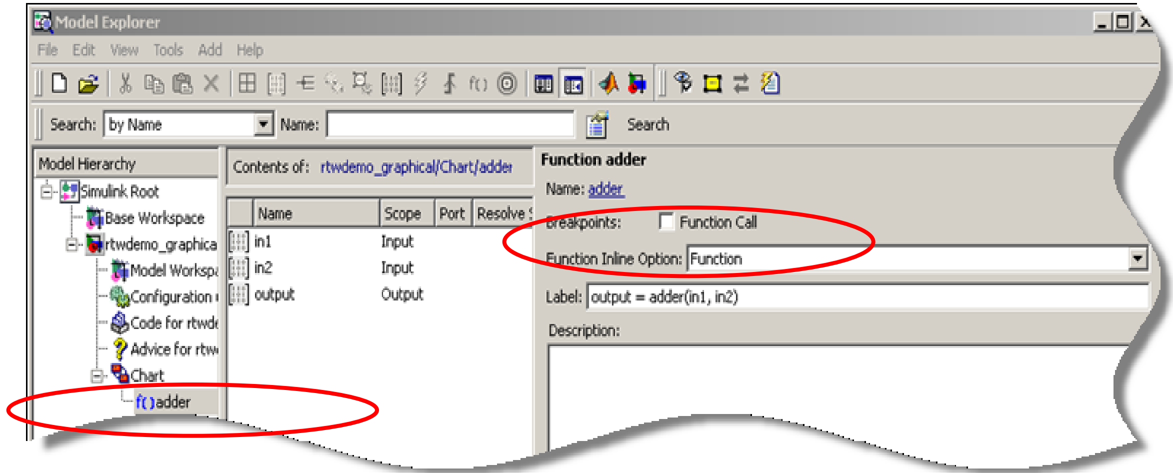
Code Pattern	Modeling Methodology
double adder (double u_1, double u_2) { return u_1 +u_2 ; } ***Description***: *Obtain functions in the generated code, that take input arguments and return an output. In most cases Simulink Subsystems do not generate function signatures which return values. However, pointers to the function outputs can be passed in as arguments. The function prototype can be controlled for the top-level of a model.*	1. Use a Stateflow graphical function and export the function globally. 2. Create a model and control the function prototype of the <model>_step function

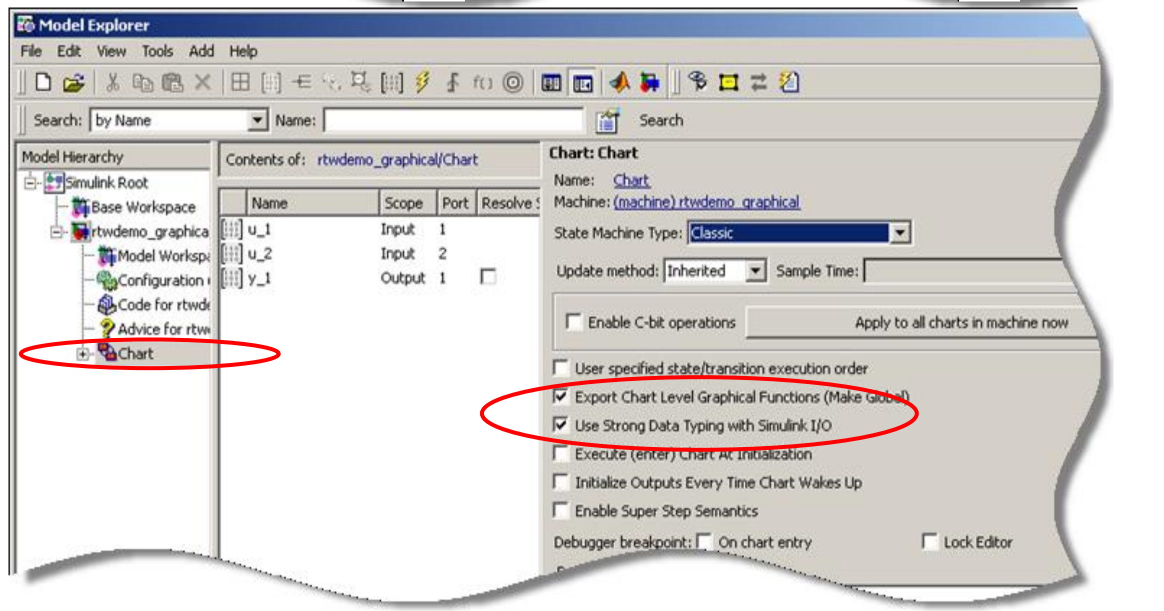
functions

Creation with Graphical Function









Generated Code

```
real_T adder(real_T sf_in1, real_T sf_in2)
{
  return sf_in1 + sf_in2;
}
```

rtwdemo_graphical.mdl

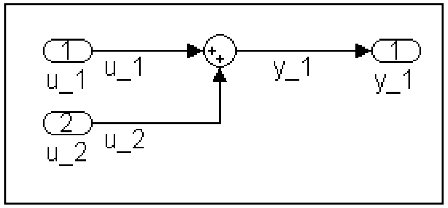
functions

Top-level function prototype control

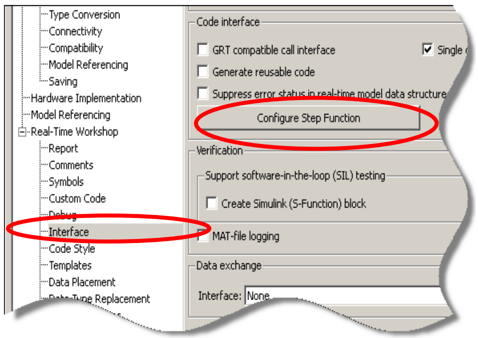
C- Code Pattern

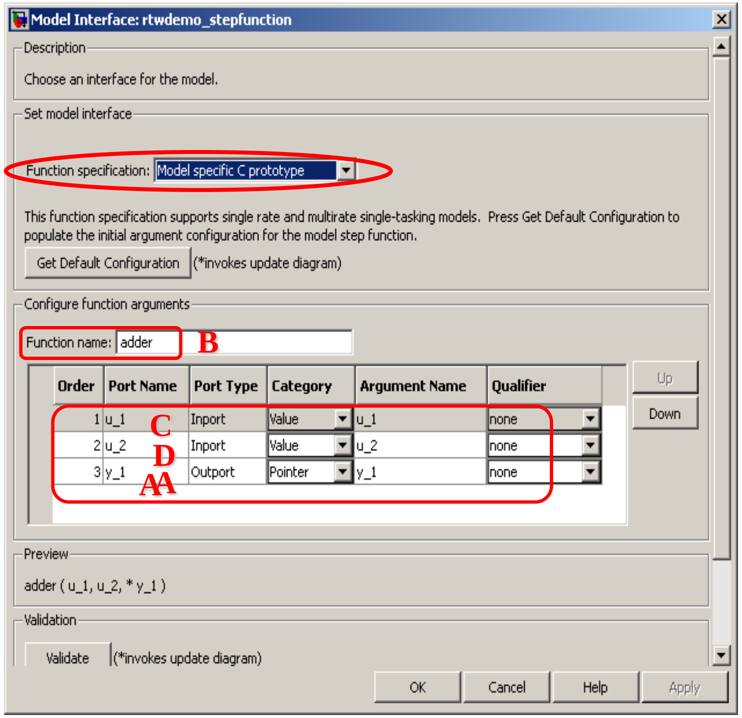
A B C D

```
double adder (double u_1, double u_2)
{
    return u_1 + u_2;
}
```



Configuration wizard (ctrl + E)





Generated Code

B C D A

```
void adder(real_T u_1, real_T u_2, real_T *y_1)
{
  (*y_1) = u_1 + u_2;
}
```

rtwdemo_stepfunction.mdl

C Code Pattern: Calling external C functions

Code Pattern, Custom Function	Modeling Methodology
``/* filename: add.h */ #ifndef _ADD_H #define _ADD_H extern double add(double,double); #endif /* EOF */```	1. Use the Legacy Code Toolbox to create an S-function
``/* filename: add.c */ #include "add.h" double add(double u_1, double u_2) { double y_1; y_1 = u_1 + u_2; return (y_1); } /* EOF */```	2. Use Stateflow Target to call external code.
Description: Integrate custom/legacy functions in the generated code.	2. Use Embedded MATLAB Function to call external code.

```
/* filename: add.h */
#ifndef _ADD_H
#define _ADD_H

extern double add(double,double);

#endif

/* EOF */
```

```
/* filename: add.c */
#include "add.h"
double add(double u_1, double u_2)
{
    double y_1;
    y_1 = u_1 + u_2;
    return (y_1);
}
/* EOF */
```

Return to Table of Contents

Calling External C functions

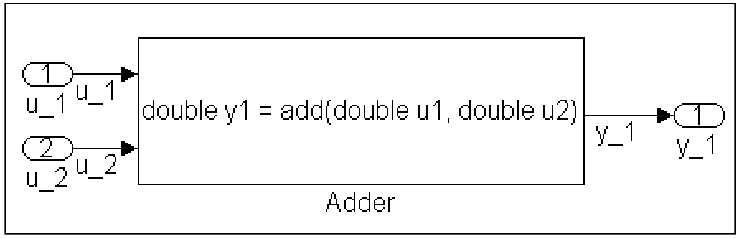
Creation

SL	SF	EML

```c
/* filename: add.h */
#ifndef _ADD_H
#define _ADD_H

extern double add(double,double);

#endif
/* EOF */
```

```c
/* filename: add.c */
#include "add.h"
double add(double u_1, double u_2)
{
   double y_1;
   y_1 = u_1 + u_2;
   return (y_1);
}
/* EOF */
```

```matlab
%% Initialize legacy code tool data structure
def = legacy_code('initialize') ;
%% Specify Source File
def.SourceFiles = {'add.c'};
%% Specify Header File
def.HeaderFiles = {'add.h'};
%% Specify the Name of the generated S-function
def.SFunctionName = 'Adder';
%% Create a c-mex file for S-function
legacy_code('sfcn_cmex_generate', def);
%% Define function signature and target the Output method
def.OutputFcnSpec = ['double y1 = add(double u1, double u2)'];
%% Compile/Mex and generate a block that can be used in simulation
legacy_code('generate_for_sim', def);
%% Create a TLC file for Code Generation
legacy_code('sfcn_tlc_generate', def);
%% Create a Masked S-function Block
legacy_code('slblock_generate', def);
```



Generated Code

<model>.h

#include "add.h"

<model>.c

y_1 = add(u_1, u_2);

rtwdemo_leg_code_model.mdl

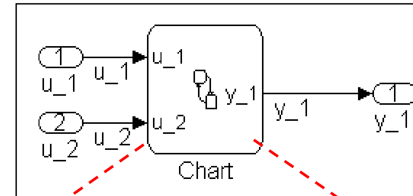
Calling External C functions

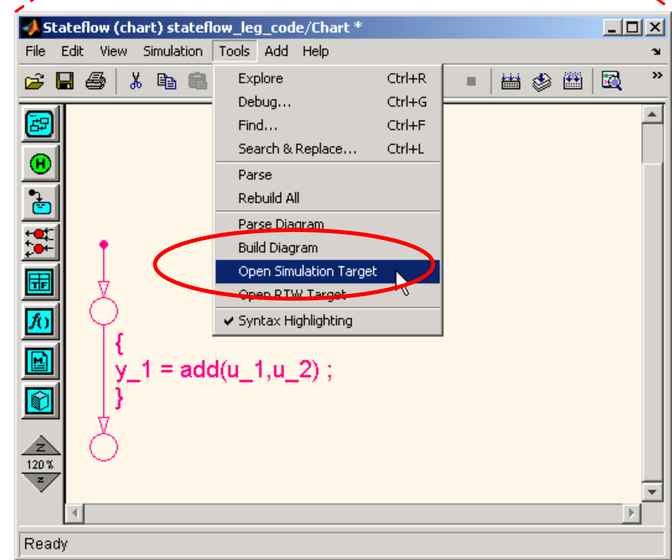
Creation

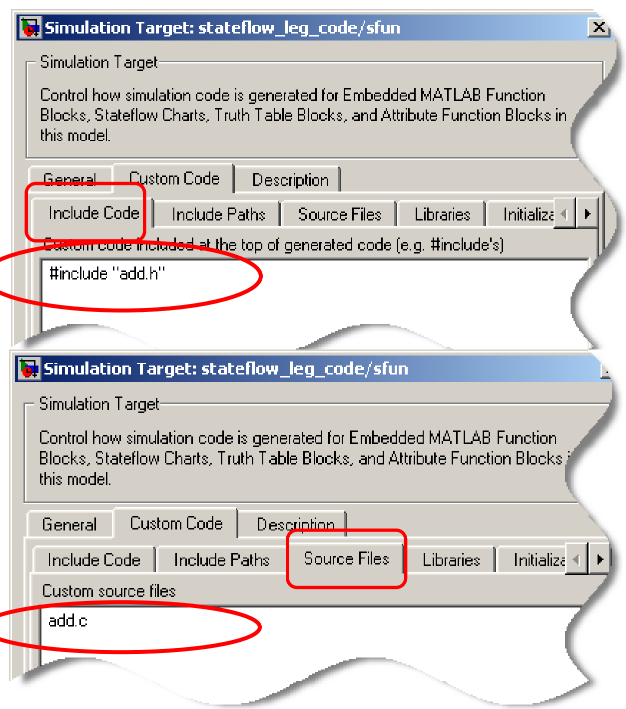
SL	SF	EML

```c
/* filename: add.h */
#ifndef _ADD_H
#define _ADD_H

extern double add(double,double);

#endif

/* EOF */
```

```c
/* filename: add.c */
#include "add.h"
double add(double u_1, double u_2)
{
   double y_1;
   y_1 = u_1 + u_2;
   return (y_1);
}
/* EOF */
```







Generated Code

<model>.h

#include "add.h"

<model>.c

y_1 = (real_T)**add**(u_1, u_2);

rtwdemo_stateflow_leg_code.mdl

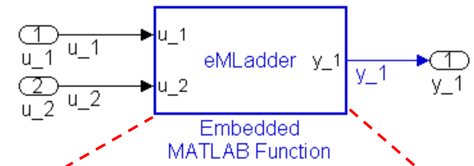
Calling External C functions

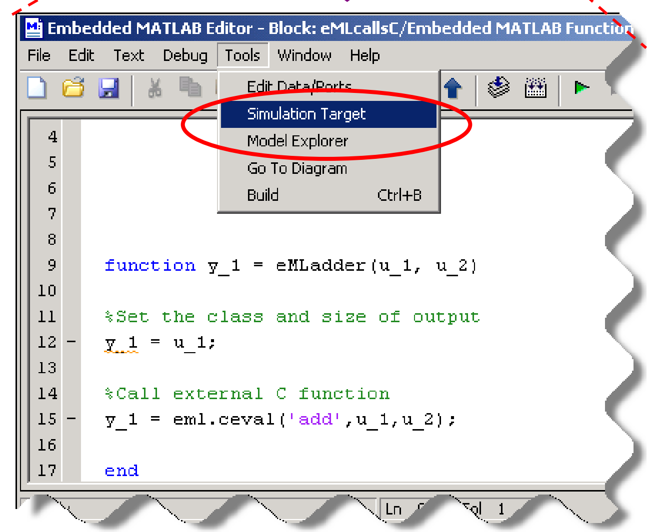
Creation

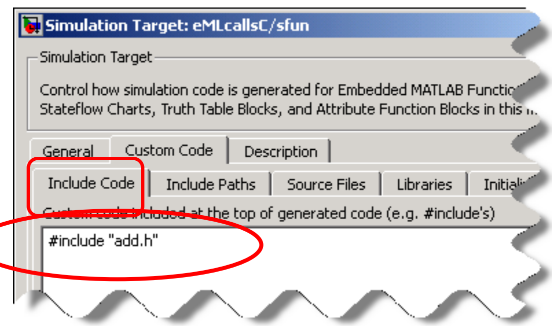
SL | SF | EML

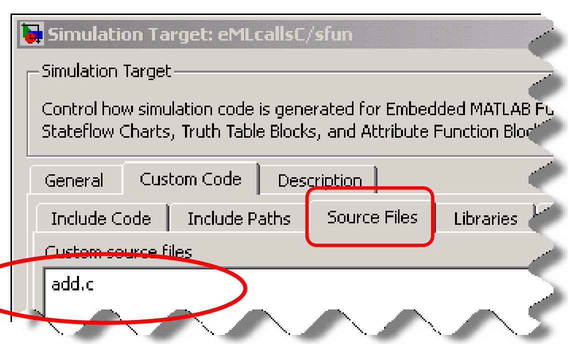
```
#ifndef _ADD_H
#define _ADD_H

#include "tmwtypes.h"
extern double add(double,double) ;

#endif

/* EOF */
```

```
/* filename: add.c */
#include "add.h"
double add(double u_1, double u_2)
{
   double y_1;
   y_1 = u_1 + u_2;
   return (y_1);
}
/* EOF */
```









Generated Code

```
{
   real_T eml_y_1;
  eml_y_1 = add(u_1, u_2);
   y_1 = eml_y_1;
 }
```

rtwdemo_eMLcallsC.mdl

C Code Pattern: #define

Code Pattern for Parameters	Modeling Methodology
#define p_1 9.8 *Description: Use #define macros for constant parameters.*	1. Create a Data Object for the parameter and use the 'Define' CSC. 2. Import the parameters from a custom header file with the ImportFromFile CSC.

Return to Table of Contents

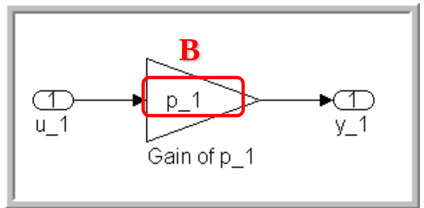
#define

Creation with Define CSC

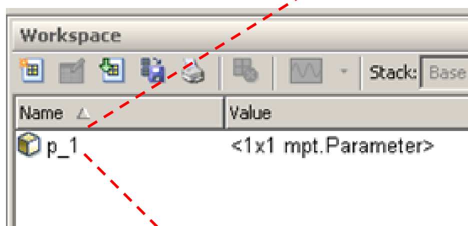
C- Code Pattern

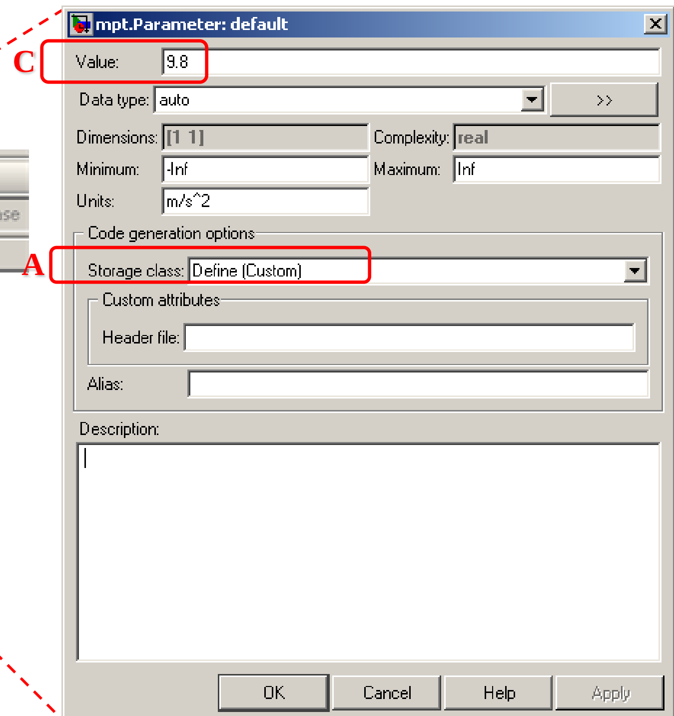
A B C

#define p_1 9.8



>> p_1 = mpt.Parameter;





Generated Code

<model>.h

A B C

#define p_1 9.8

rtwdemo_pound_define.mdl

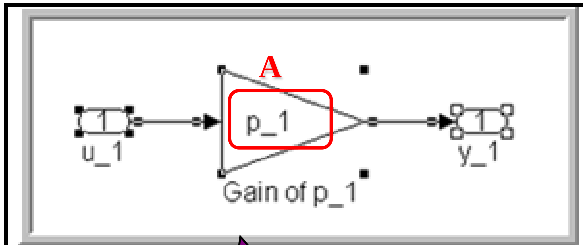
#define

Creation with ImportFromFile CSC

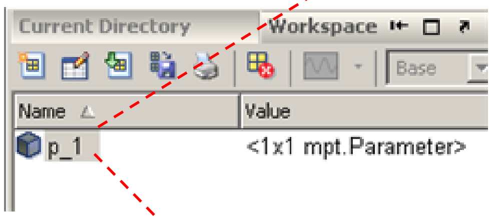
C- Code Pattern

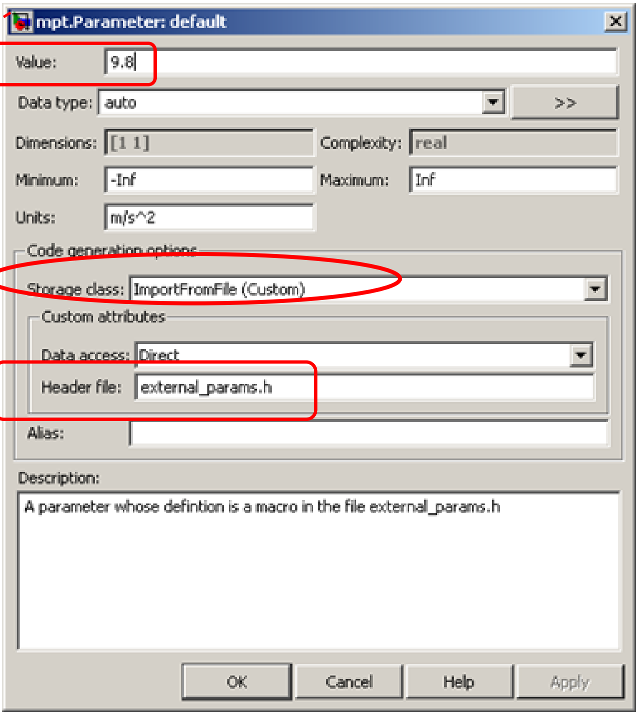
A B

```
#define p_1 9.8
```




```
>> p_1 = mpt.Parameter
```





Generated Code

<model>_private.h

C

```
#include "external_params.h"
```

Custom Header File
external_params.h

A B

```
#ifndef __EXTERNAL_PARAMS__
#define __EXTERNAL_PARAMS__

#define p_1    9.8
#define p_2    1.633

#endif
```

rtwdemo_pound_define2.mdl

C Code Pattern: typedef

Code Pattern	Modeling Methodology
`typedef double float_64 ;` ***Description***: *Obtain and use a typedef data type in the generated code.*	Use Simulink AliasType data object.

Return to Table of Contents

typedef

Creation with Data Object

C- Code Pattern

A B

typedef double float_64;

Step 1:

B

>> float_64 = Simulink.AliasType

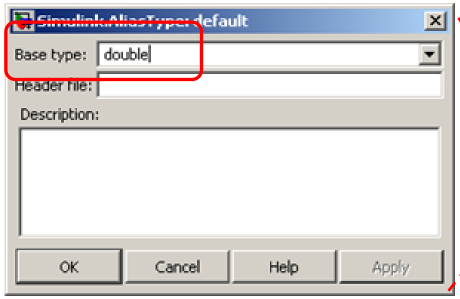
Step 2:

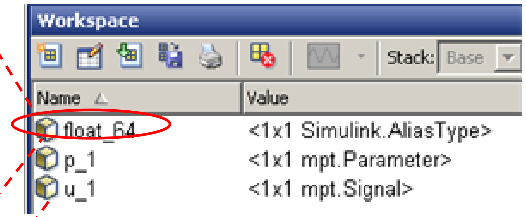
Create Simulink Data objects to signals/Parameters if necessary

Step 3:

A

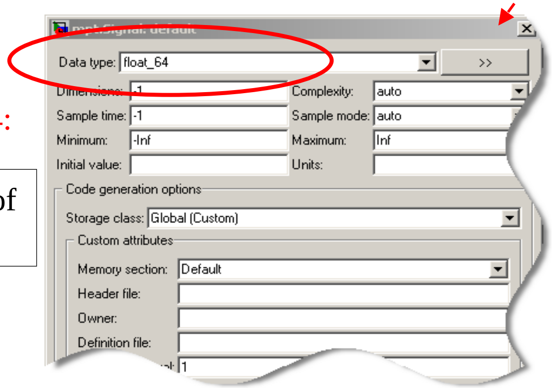
Specify Base Type





Step 4:

Specify usage of alias data type



rtwdemo_get_type_def.mdl

Generated Code

<model>_types.h

A B

typedef real_T float_64;

NOTE: real_T is the Real-Time Workshop Embedded Coder typedef for double

C Code Pattern: Structures

Code Pattern, Parameters	Modeling Methodology
typedef struct{ double p_1; double p_2; double p_3; }my_struct_type; my_struct_type my_struct={1.0,2.0,3.0}; ***Description:*** *Define a data-type structure and create an object to that data-type to initialize parameters.*	Use Simulink Data-Objects with Struct CSC (Custom Storage Class)

Return to Table of Contents

Structure typedef for Parameters

Creation with Struct CSC

C- Code Pattern, Parameters

```
typedef struct{    A
        double p_1;
        double p_2;
        double p_3;
}my_struct_type;
                    B
my_struct_type my_struct ={1.0,2.0,3.0};
```

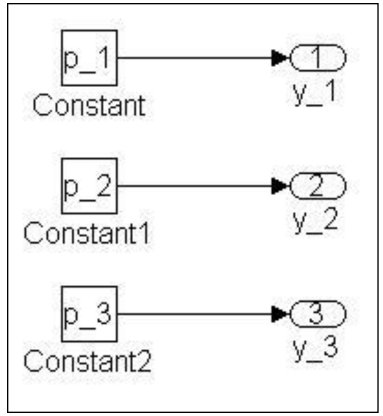
Generated Code, Parameters

<model>_types.h

```
A typedef struct my_struct_tag {
    real_T p_1;
    real_T p_2;
    real_T p_3;
  } my_struct_type;
```

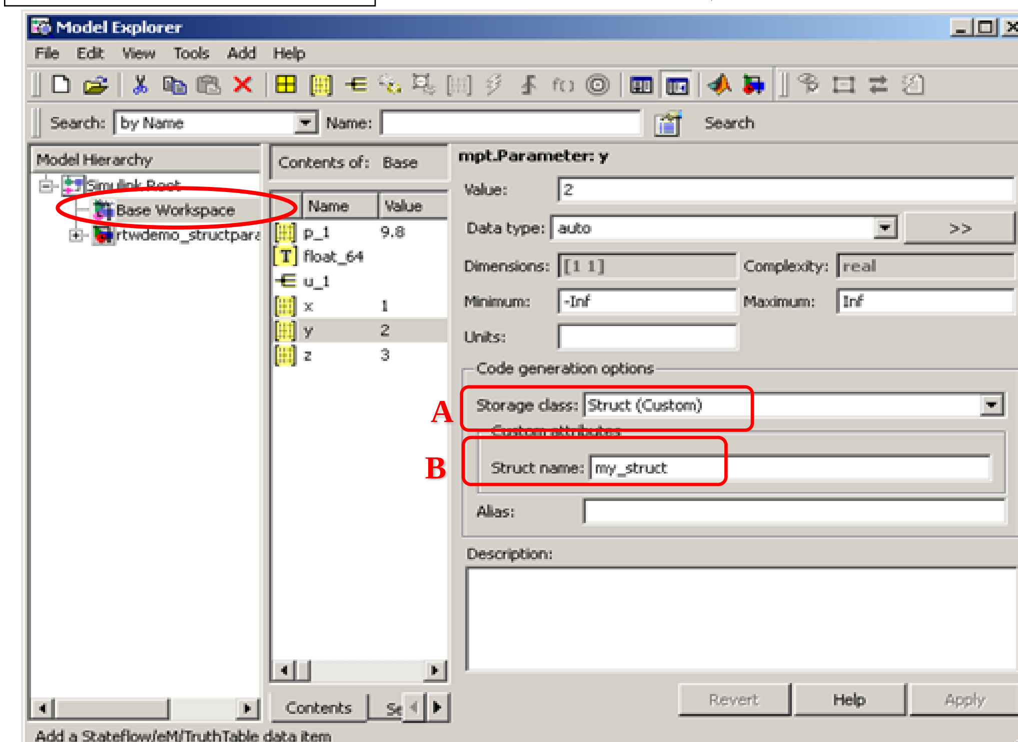
<model>.c

```
                              B
  my_struct_type my_struct = {
    /* p_1 */
    1.0,

    /* p_2 */
    2.0,

    /* p_3 */
    3.0
  };
```



Create Data Objects for Parameters in the Workspace

```
>> p_1 = mpt.Parameter ;
>> p_2 = mpt.Parameter ;
>> p_3 = mpt.Parameter ;
```



rtwdemo_structparam.mdl

C Code Pattern: Structure typedef for Signals

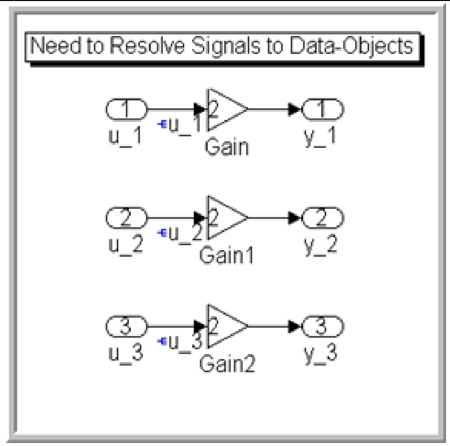
Code Pattern, Signals	Modeling Methodology
typedef struct { double u_1; double u_2; double u_3; }MySignals; ***Description**: Define a structure containing signals.*	1. Use Simulink Data-Objects with Struct CSC (Custom Storage Class) 2. Use Simulink Non-Virtual Bus Objects.

Structure typedef for Signals

Creation with CSC

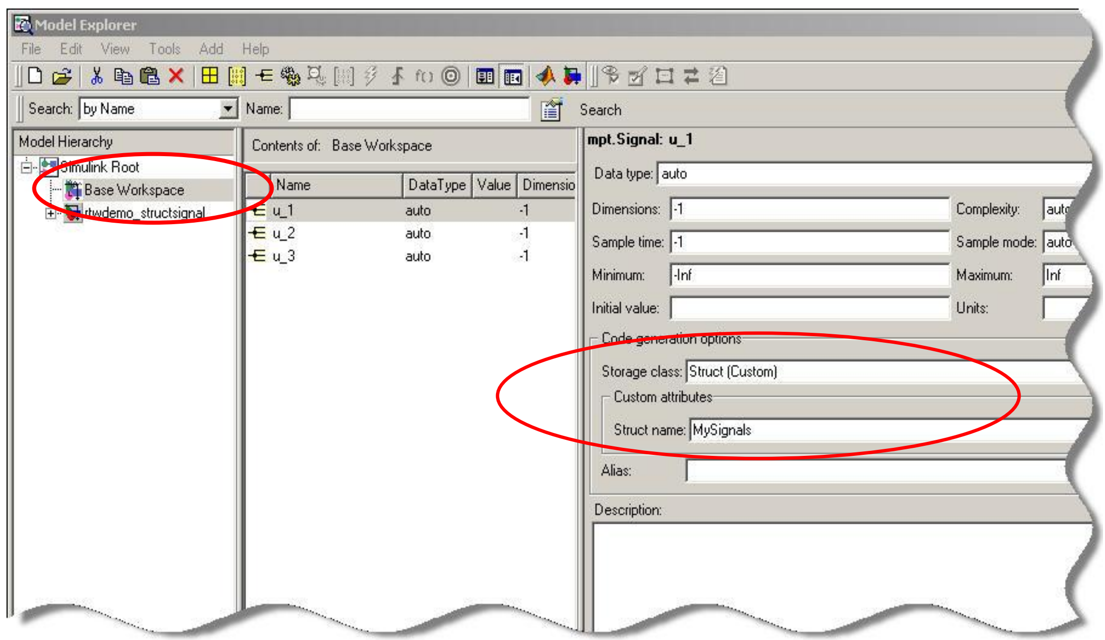
C- Code Pattern, Signals

```
typedef struct {
        double u_1;
        double u_2;
        double u_3;
}MySignals;
```



Create Simulink Data Objects

```
>> u_1 = mpt.Signal;
>> u_2 = mpt.Signal;
>> u_3 = mpt.Signal ;
```



Generated Code, Parameters

<model>_types.h

```
typedef struct MySignals_tag {
  real_T u_1;
  real_T u_2;
  real_T u_3;
} MySignals_type;
```

NOTE : real_T is a Real-Time Workshop Embedded Coder Typedef for double.

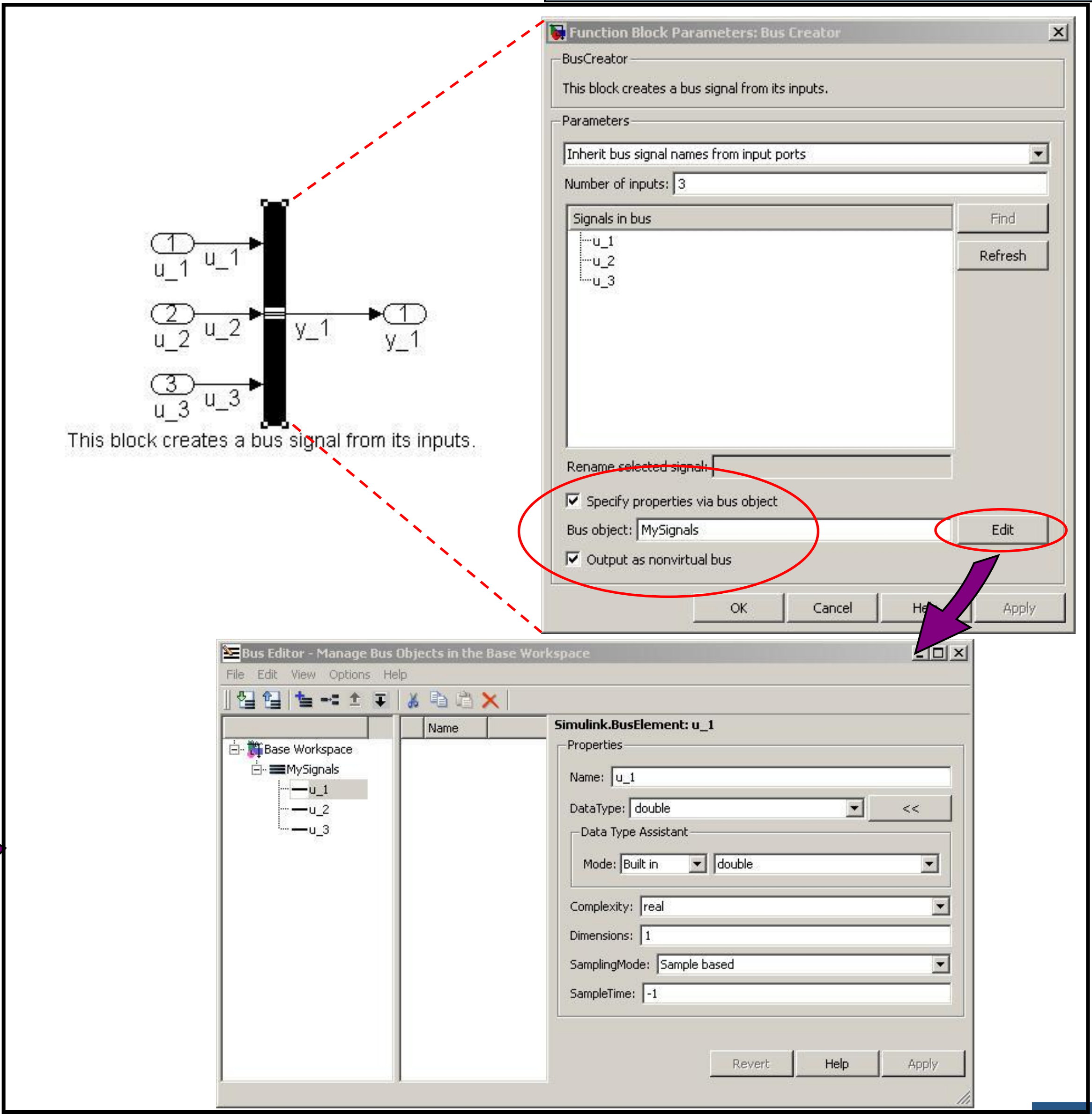
rtwdemo_structsignal.mdl

Structure typedef for Signals

Creation with Bus Objects

C- Code Pattern, Signals

```
typedef struct {
        double u_1;
        double u_2;
        double u_3;
}MySignals;
```



Generated Code, Signals

<model>_types.h

```
typedef struct {
  real_T u_1;
  real_T u_2;
  real_T u_3;
} MySignals;
```

rtwdemo_structsignal2.mdl

C Code Pattern: Nested Structures

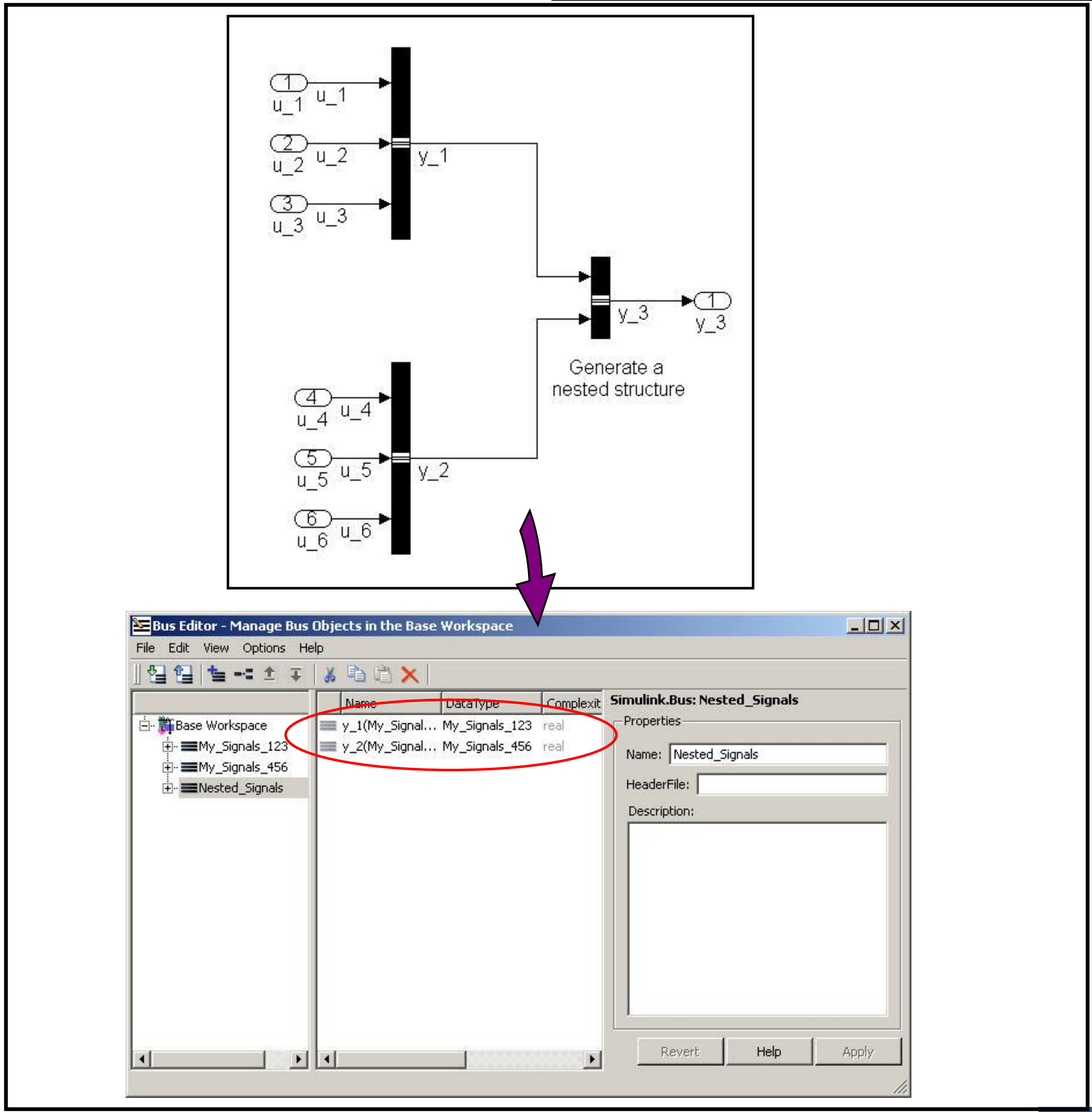
Code Pattern, Signals	Modeling Methodology
<pre>typedef struct { real_T u_1; real_T u_2; real_T u_3; }My_Signals_123; typedef struct { real_T u_4; real_T u_5; real_T u_6; }My_Signals_456; typedef struct { My_Signals_123 y_1; My_Signals_456 y_2; } Nested_Signals;</pre> *Description: Define a nested structure.*	Use Simulink Non-Virtual Bus Objects.

Structure typedef for Signals

Creation with Bus Objects

C- Code Pattern, Signals

```
typedef struct {
        real_T u_1;
        real_T u_2;
        real_T u_3;
}My_Signals_123;

typedef struct {
        real_T u_4;
        real_T u_5;
        real_T u_6;
}My_Signals_456;

typedef struct {
  My_Signals_123 y_1;
  My_Signals_456 y_2;
} Nested_Signals;
```



rtwdemo_nestedstructure.mdl

continued ...

Generated Code, Signals

```
#ifndef _DEFINED_TYPEDEF_FOR_My_Signals_123_
#define _DEFINED_TYPEDEF_FOR_My_Signals_123_

typedef struct {
  real_T u_1;
  real_T u_2;
  real_T u_3;
} My_Signals_123;

#endif

#ifndef _DEFINED_TYPEDEF_FOR_My_Signals_456_
#define _DEFINED_TYPEDEF_FOR_My_Signals_456_

typedef struct {
  real_T u_4;
  real_T u_5;
  real_T u_6;
} My_Signals_456;

#endif

#ifndef _DEFINED_TYPEDEF_FOR_Nested_Signals_
#define _DEFINED_TYPEDEF_FOR_Nested_Signals_

typedef struct {
  My_Signals_123 y_1;
  My_Signals_456 y_2;
} Nested_Signals;

#endif
```

C Code Pattern: Bit-Fields

Code Pattern	Modeling Methodology
typedef struct { unsigned int p_1 : 1; unsigned int p_2 : 1; unsigned int p_3 : 1; }My_Struct_type; ***Description***: *Define a structure containing bit fields.*	Use Simulink Data-Objects with Bitfield CSC (Custom Storage Class)

Return to Table of Contents

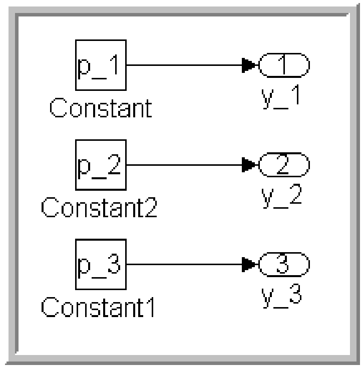
Bit Fields

Creation with Bitfield CSC

C- Code Pattern

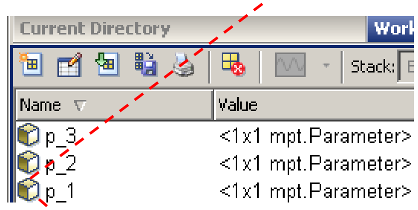
```
typedef struct {
        unsigned int p_1 : 1;
        unsigned int p_2 : 1;
        unsigned int p_3 : 1;
}My_Struct_type;
```

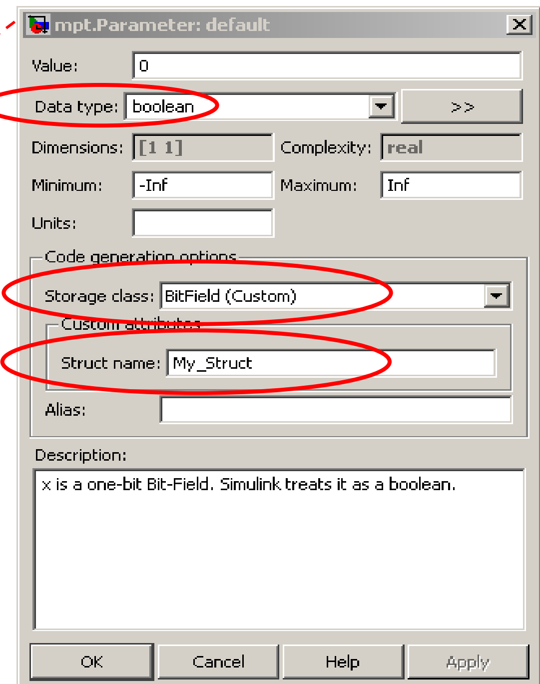
rtwdemo_bitfield.mdl



Create Data Objects for Parameters in the Workspace

```
>> p_1 = mpt.Parameter ;
>> p_2 = mpt.Parameter ;
>> p_3 = mpt.Parameter ;
```





Generated Code

<model>_types.h

```
typedef struct My_Struct_tag {
  uint_T p_1 : 1;
  uint_T p_2 : 1;
  uint_T p_3 : 1;
} My_Struct_type;
```

NOTE: unit_T is the Real-Time Workshop Embedded Coder typedef for unsigned int

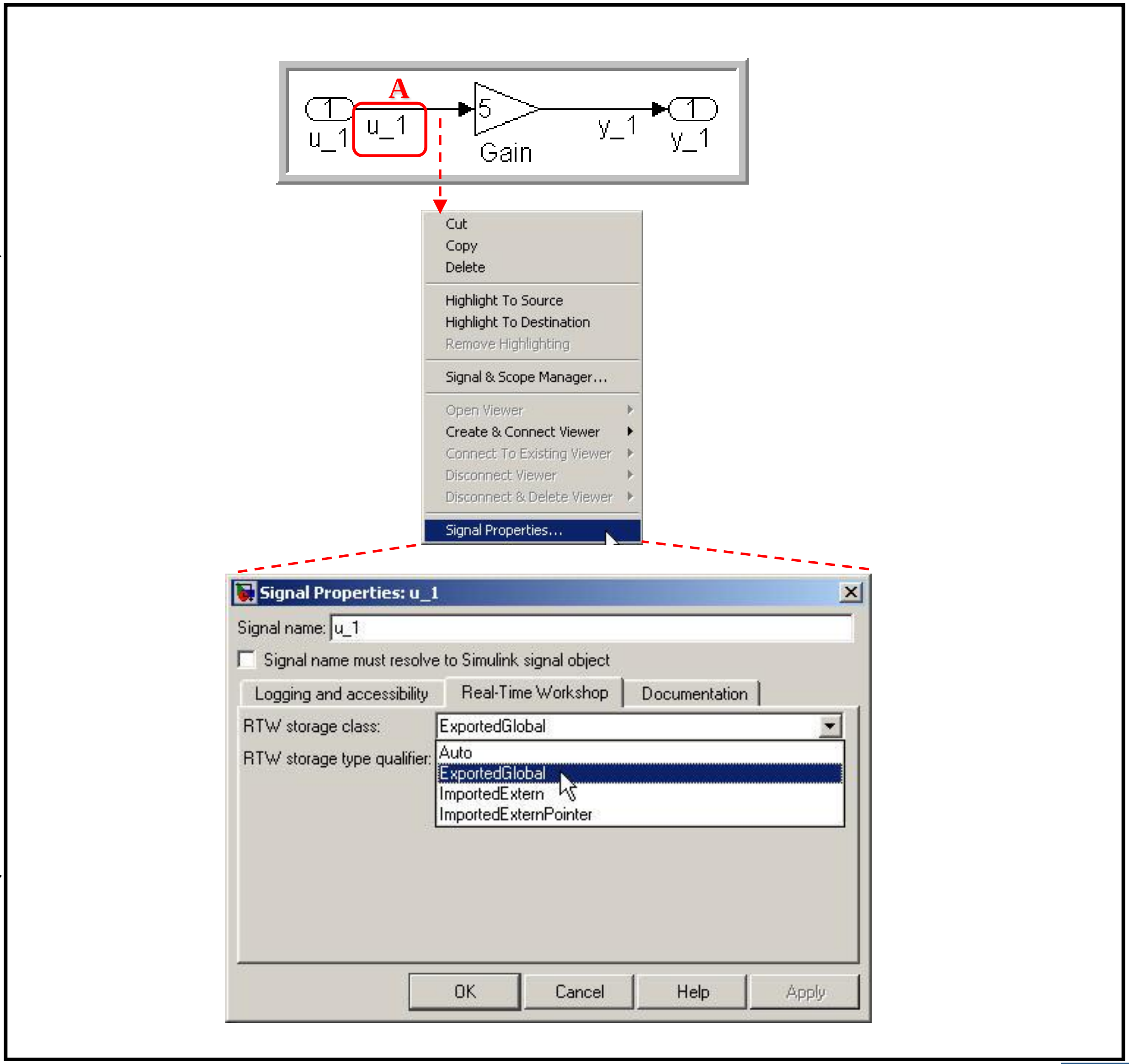
C Code Pattern: Arrays

Code Pattern, Signal Variables	Code Pattern, Parameter Constants
`int u_1[5];` `int y_1[5];` `for (inx=0;inx<5;inx++)` `{` `y_1[inx] = 5 * u_1[inx] ;` `}` Description: *The aim is to obtain arrays such as the ones above in the generated code. u_1 and y_1 are signal arrays with integer data type. An example operation on arrays is also shown.*	`int params[5]= {1,2,3,4,5} ;` Description: *params is a constant array of parameters.*

C- Code Pattern, Signal

A B

```
int u_1 [5];
int y_1  [5];

for (inx=1;inx<5;inx++){
y_1[inx] = 5 * u_1[inx] ;
}
```



Generated Code, Signal

A B

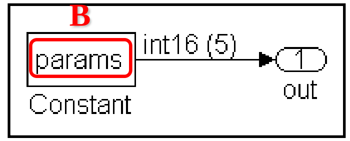
```
int16_T u_1[5];
int16_T y_1[5];

for (i = 0; i < 5; i++) {
     y_1[i] = (int16_T)(5 * u_1[i]);
   }
```

rtwdemo_array_signals_sdo.mdl
rtwdemo_array_signals.mdl

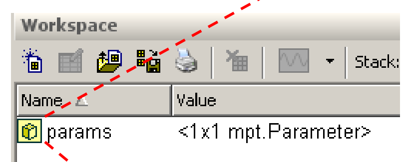
C- Code Pattern, Parameter

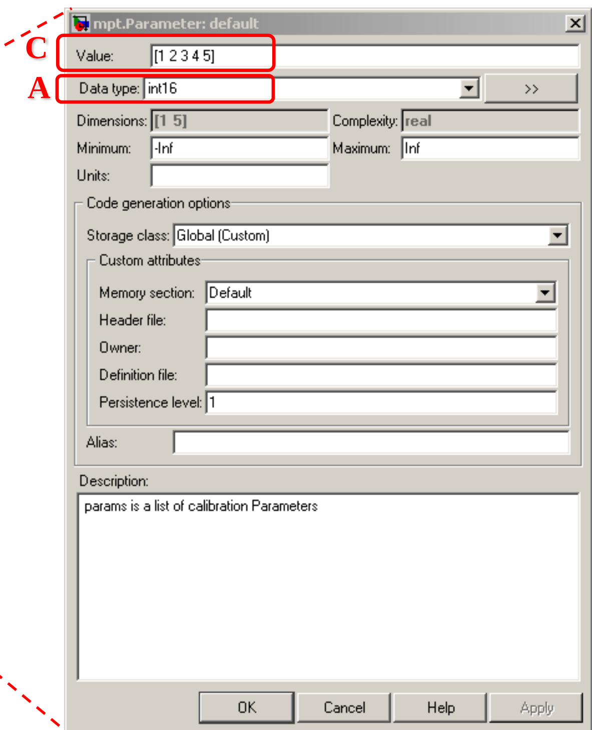
A B C

int params [5]={1,2,3,4,5} ;



>> params = [1 2 3 4 5];





Generated Code, Parameter

A B C

int16_T params [5] = { 1, 2, 3, 4, 5 } ;

rtwdemo_array_params_sdo.mdl
rtwdemo_array_params.mdl

C Code Pattern: Pointers

<table>
<tr><th>Code Pattern</th><th>Modeling Methodology</th></tr>
<tr><td rowspan="2"><code>extern double *u_1 ;</code>

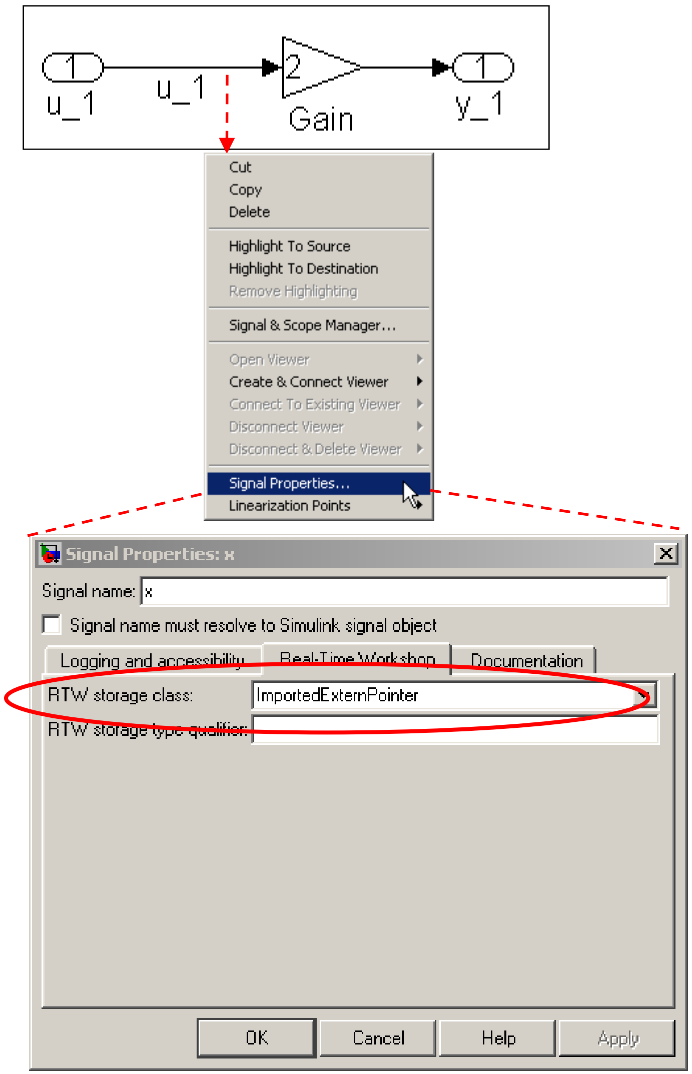
Description: Import a pointer to a signal/parameter in the generated code.</td><td>1. Use the Imported Extern Pointer Storage Class for signals. Eliminates need for data objects.</td></tr>
<tr><td>2. Create a data object for the signal/parameter and use the imported Extern Pointer Storage Class.</td></tr>
</table>

Return to Table of Contents

Creation with Imported Extern Storage Class

C- Code Pattern

```
extern double *u_1 ;
```



Generated Code

<model>_private.h

```
extern real_T *u_1;
```

NOTE: real_T is the Real-Time Workshop Embedded Coder typedef for double

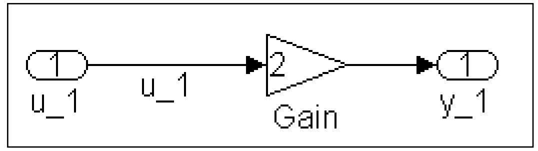
rtwdemo_pointer_signal.mdl

Creation with Data Objects

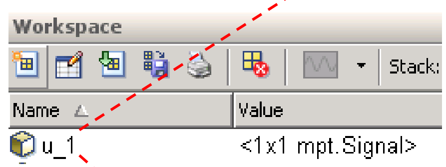
C- Code Pattern

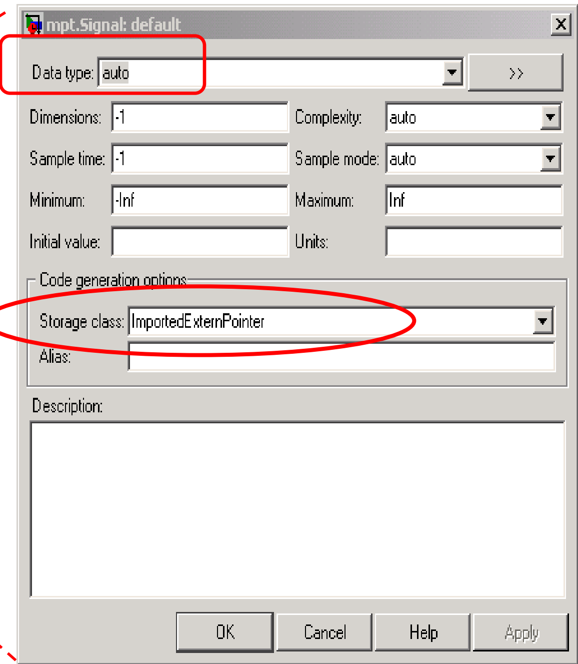
A

extern double *u_1 ;




```
>> u_1 = mpt.Signal
```





Generated Code

<model>_private.h

A

extern real_T *u_1;

NOTE: real_T is the
Real-Time Workshop Embedded Coder
typedef for double

rtwdemo_pointer_signal_sdo.mdl