Realize Your Product Promise®

Model-Based Design
with
SCADE Suite®

Training Activities

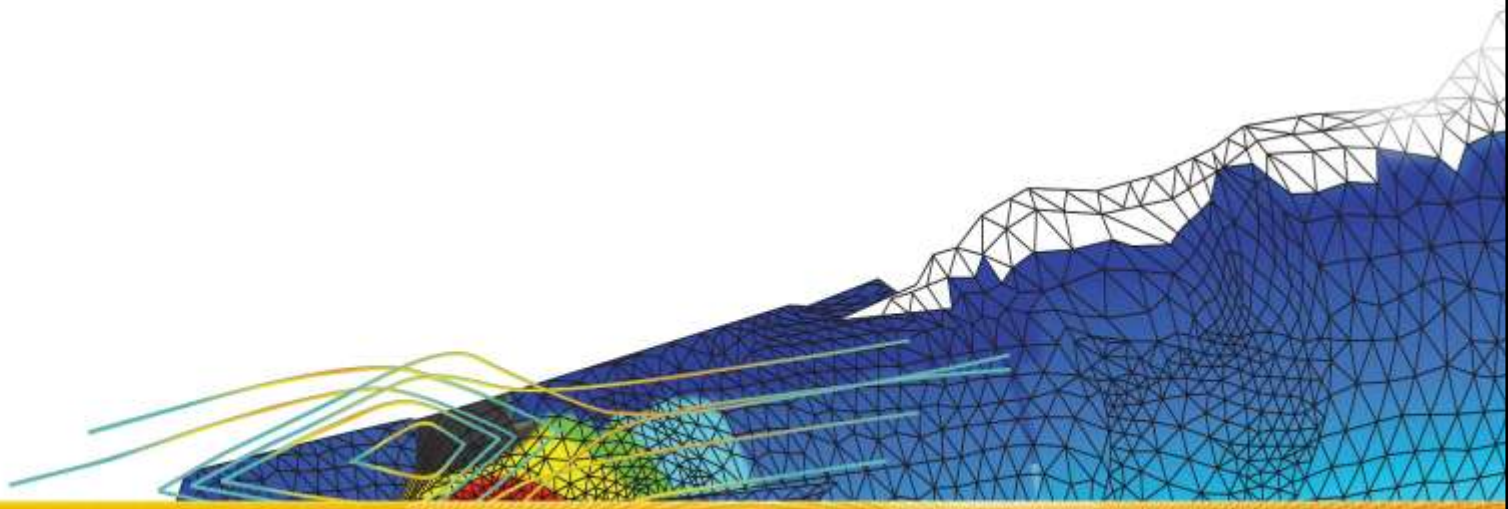
Day 3

ANSYS

ANSYS

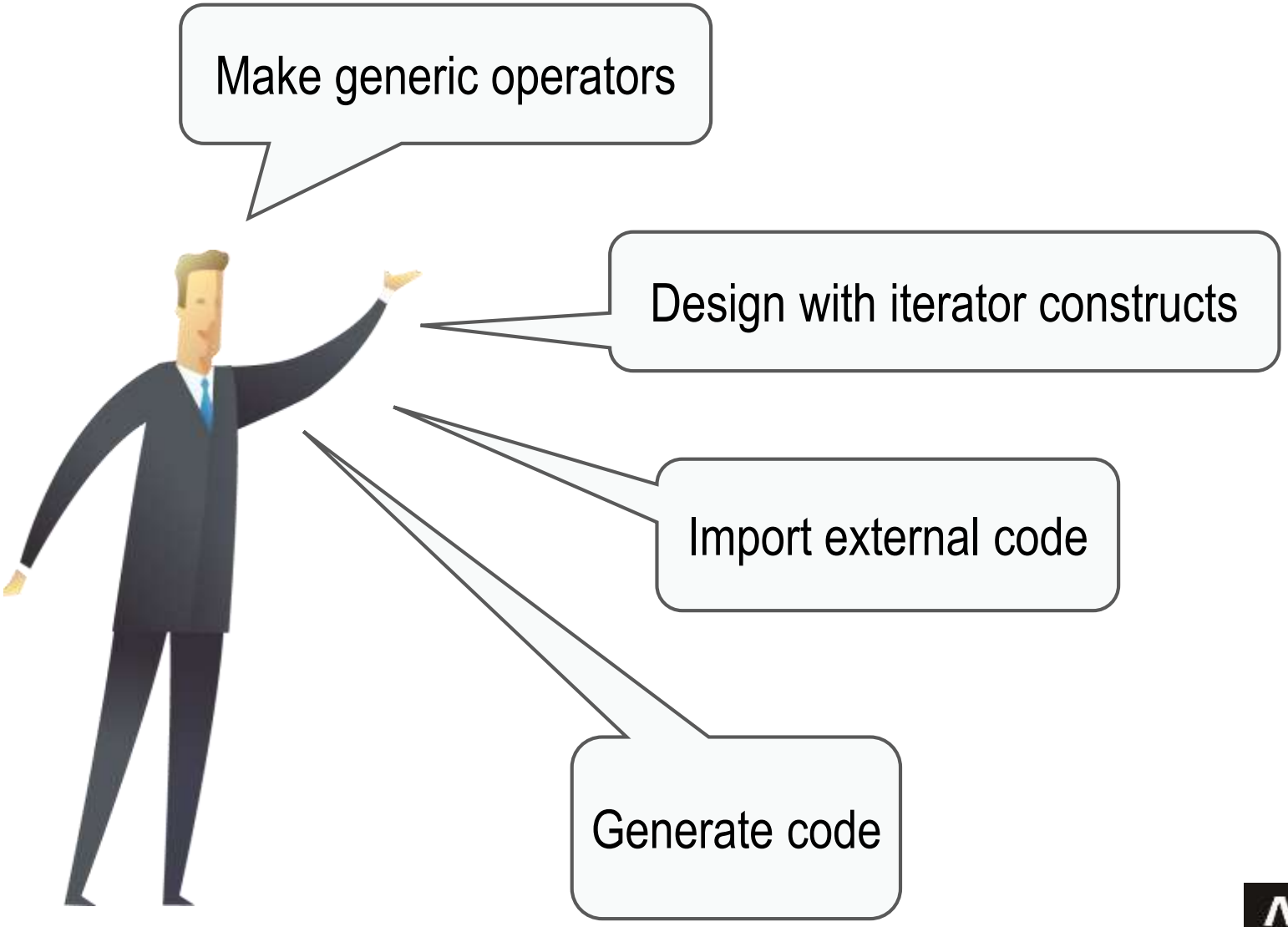
Model-Based Design with SCADE Suite

Lab Support Day 3



Lab Objectives

Lab



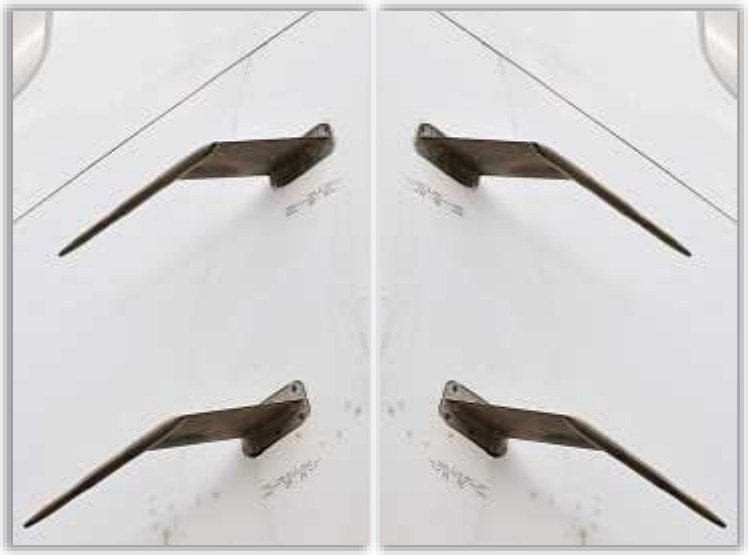
Prerequisites

Lab Prerequisite

Safety Critical Systems interact with their physical environment.

They rely on sensors to observe their physical environment, and on actuators to act on it.

A failure of a sensor or an actuator is a Single Point of Failure: so they need to be redundant



Redundant Pitot sensors on a B737

Prerequisites

Wikipedia

A pitot tube is a pressure measurement instrument used to measure fluid flow velocity.

It is widely used to determine the airspeed of an aircraft, water speed of a boat, and to measure liquid, air and gas velocities in industrial applications.

The pitot tube measures the local velocity at a given point in the flow stream and not the average velocity in the pipe or conduit.

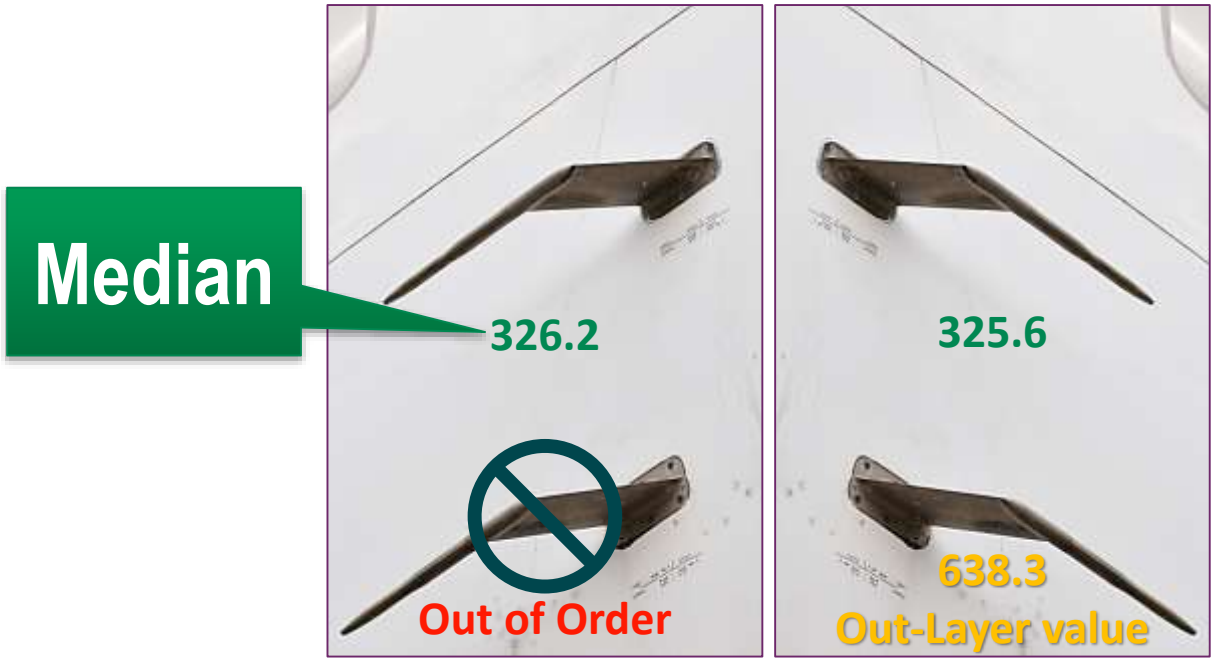
Lab Prerequisite

Prerequisites

Having redundant sensors means that from a range of values one final value has to be determined.

A naive way to determine this value is to use the mean of valid values.

A smarter way to do this is to use the median.



Lab Prerequisite

Prerequisites

Lab Prerequisite

In statistics and probability theory, the median is the numerical value separating the higher half of a data sample, a population, or a probability distribution, from the lower half.

The median of a finite list of numbers can be found by arranging all the observations from the lowest value to the highest value and picking the middle one:

The median of {3, 3, 5, 9, 11} is 5

If there is an even number of observations, then there is no single middle value. The median is usually defined to be the mean of the two middle values, which corresponds to interpreting the median as the fully trimmed mid-range:

The median of {3, 5, 7, 9} is (5 + 7) / 2 = 6

ANSYS

Lab Objectives

The aim of this one day lab is to develop a median library operator (users will implement the Torben's method in the Lab 9).

You will work on a set of values (any size).

To be able to tell if a value is above or below another one, these values need to have a sorting relation :

>

In Scade, this means that these values are numeric.

Lab Prerequisite

Lab Objectives

Lab Prerequisite

The Torben's method

This method was created by Torben Mogensen.

It is not the fastest way of finding a median, however it does not modify the input array when looking for the median.

It becomes extremely powerful when the number of elements to consider becomes large, and when copying the input array may cause enormous overheads.

For read-only input size sets of several hundred of megabytes, it is the solution of choice, as it accesses elements sequentially and not randomly.

Be aware that it needs to read the array several times, though: a first pass is only looking for min and max values, further passes go through the array and come out with the median.

Lab 1

Lab 1: Requirement

Lab

Objective:

Prepare the design environment and construct the simulation array.

Requirements:

Time: 5 min

Create a new SCADE Suite project:

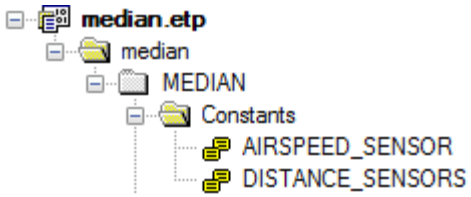
- **Project Name**: `median`
- **Package Name**: `MEDIAN`
- **No library**

Create two environment constants, under the `MEDIAN` package, to use as:

```
AIRSPEED_SENSORS (type: float64^4) =[326.2, 325.6, 638.3, 323.9]

DISTANCE_SENSORS (type: uint8^6): =[45,49,35,33,44,43]
```

ANSYS

Lab 1: Solution



Constant	Type	Value
AIRSPEED_SENS...	float64 ^4	
0	float64	326.2
1	float64	325.6
2	float64	638.3
3	float64	323.9
DISTANCE_SENS...	uint8 ^6	[45, 49, 35, 33, 44, 43]

Lab Solution

Lab 2

Lab 2: Use Array Access Operators

Lab

Objective:

Create the main operator and use array access operators.

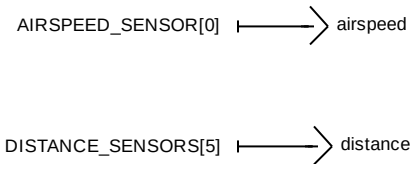
Time: 10 min

Requirements:

Create a median operator, under MEDIAN package returning:

```
airspeed = airspeed_sensors[0]
distance = distance_sensors[5]
```

Name	Kind	Type
airspeed	output	float64
distance	output	uint8



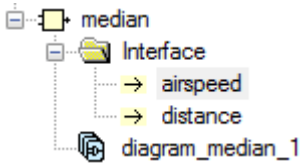
Generate the code, with KCG configuration (KCGAda for Ada target)

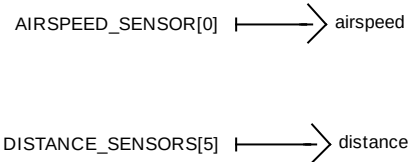
Observe the generated code and dimension matching.

Lab 2: Solution (C Code)

Lab Solution

Constant	Type	Value
AIRSPEED_SENS...	float64 ^4	
0	float64	326.2
1	float64	325.6
2	float64	638.3
3	float64	323.9
DISTANCE_SENS...	uint8 ^6	[45, 49, 35, 33, 44, 43]



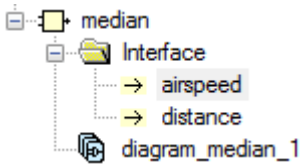


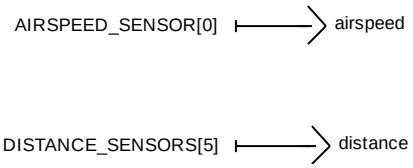

```c
#include "kcg_consts.h"

/* MEDIAN::DISTANCE_SENSORS*/
const array_uint8_6 DISTANCE_SENSORS= { kcg_lit_uint8(45), kcg_lit_uint8(49),
    kcg_lit_uint8(35), kcg_lit_uint8(33), kcg_lit_uint8(44), kcg_lit_uint8(43) };


/* MEDIAN::AIRSPEED_SENSORS*/
const array_float64_4 airspeed_sensors = { kcg_lit_float64(326.2),
    kcg_lit_float64(325.6), kcg_lit_float64(638.3), kcg_lit_float64(323,9) };
```

Lab 2: Solution (Ada)

Ada

Constant	Type	Value
AIRSPEED_SENS...	float64 ^4	
0	float64	326.2
1	float64	325.6
2	float64	638.3
3	float64	323.9
DISTANCE_SENS...	uint8 ^6	[45, 49, 35, 33, 44, 43]





Ada

```
with Kcg_Types;
with Kcg_Config;

--# inherit Kcg_Types, Kcg_Config;
package MEDIAN
-- MEDIAN::
is
  -- MEDIAN::DISTANCE_SENSORS/
  DISTANCE_SENSORS : constant Kcg_Types.Uint8_Range_0_5 :=
     Kcg_Types.Uint8_Range_0_5'(45, 49, 35, 33, 44, 43);

  -- MEDIAN::AIRSPEED_SENSORS/
  AIRSPEED_SENSORS : constant Kcg_Types.Float64_Range_0_3 :=
     Kcg_Types.Float64_Range_0_3'(326.2, 325.6, 638.3, 323.9);

  -- MEDIAN::median/
  procedure median_1(
    -- _L1/, airspeed/
    airspeed : out Kcg_Config.Kcg_Float64;
    -- _L2/, distance/
    distance : out Kcg_Config.Kcg_Uint8);
end MEDIAN;
```

Lab 3

Lab 3: Use a Generic Function

Lab

Objective:

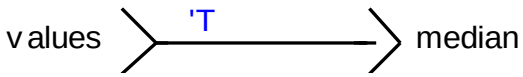
Create a generic function

Requirements:

Update the `median` function to be a generic function

Time: 10 min

Name	Kind	Type
`value`	input	'T
`median`	output	'T

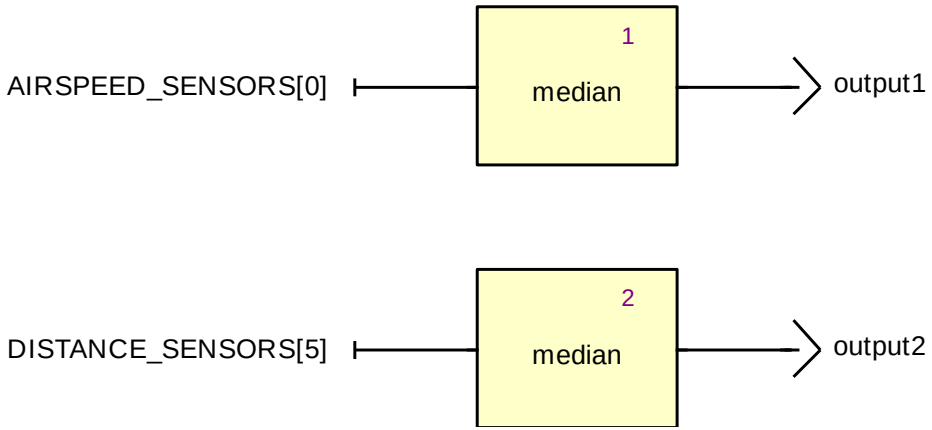


Lab 3: Use a Generic Function

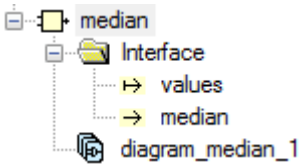
Lab

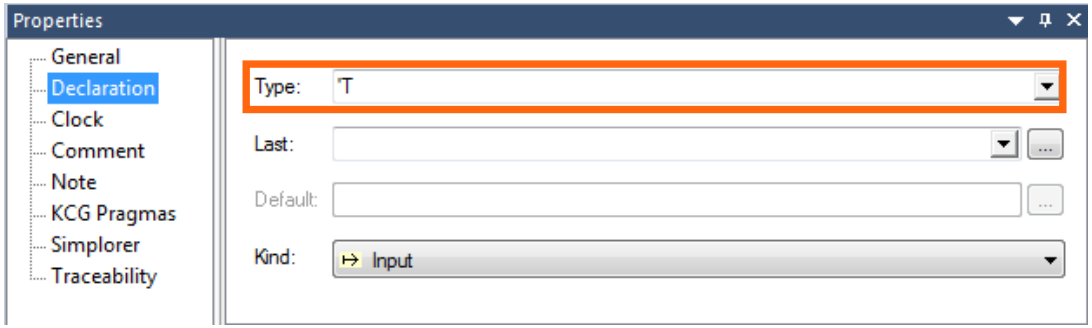
Create a `test` operator to simulate several median instances

Name	Kind	Type
`output1`	ouput	float64
`output2`	output	uint8



Lab 3: Solution



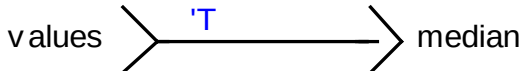


Remove one output from the median operator and transform the other output to be generic

Change the output's name in the median

Add one input, named values, with the same generic type of the output

Open the design of the operator and connect its values and median

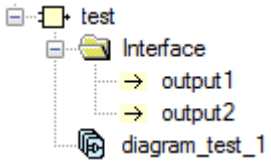


Lab Solution

Lab 3: Solution

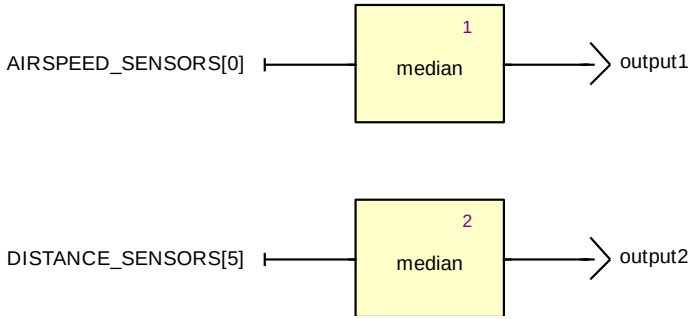
Create a “test” operator to simulate several median instances, with two outputs:

- output1: float64
- output2: uint8



Instantiate twice the median operator to recover:

- AIRSPEED_SENSORS[0]
- DISTANCE_SENSORS [5]



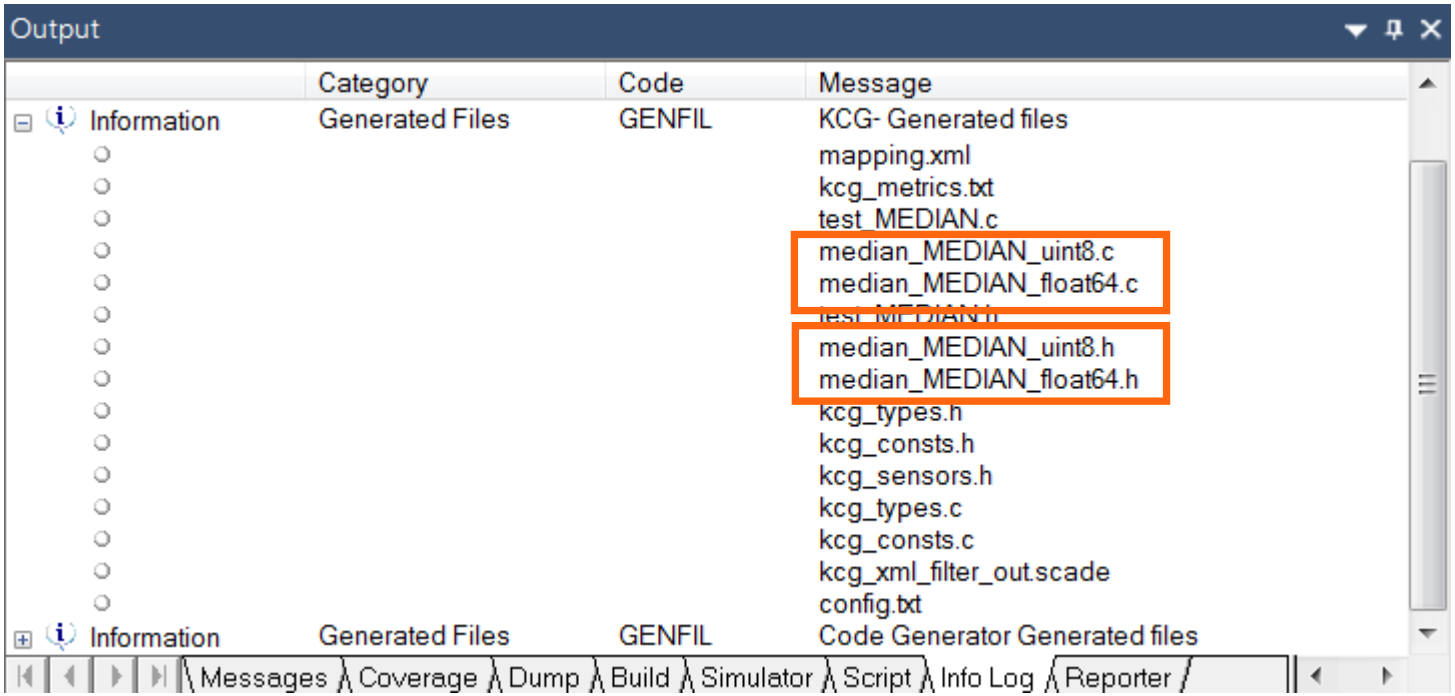
Lab Solution

Lab 3: Solution (C Code)

Lab Solution

Generate the C code.

Observe the generated code: for each median operator instance, files are generated.



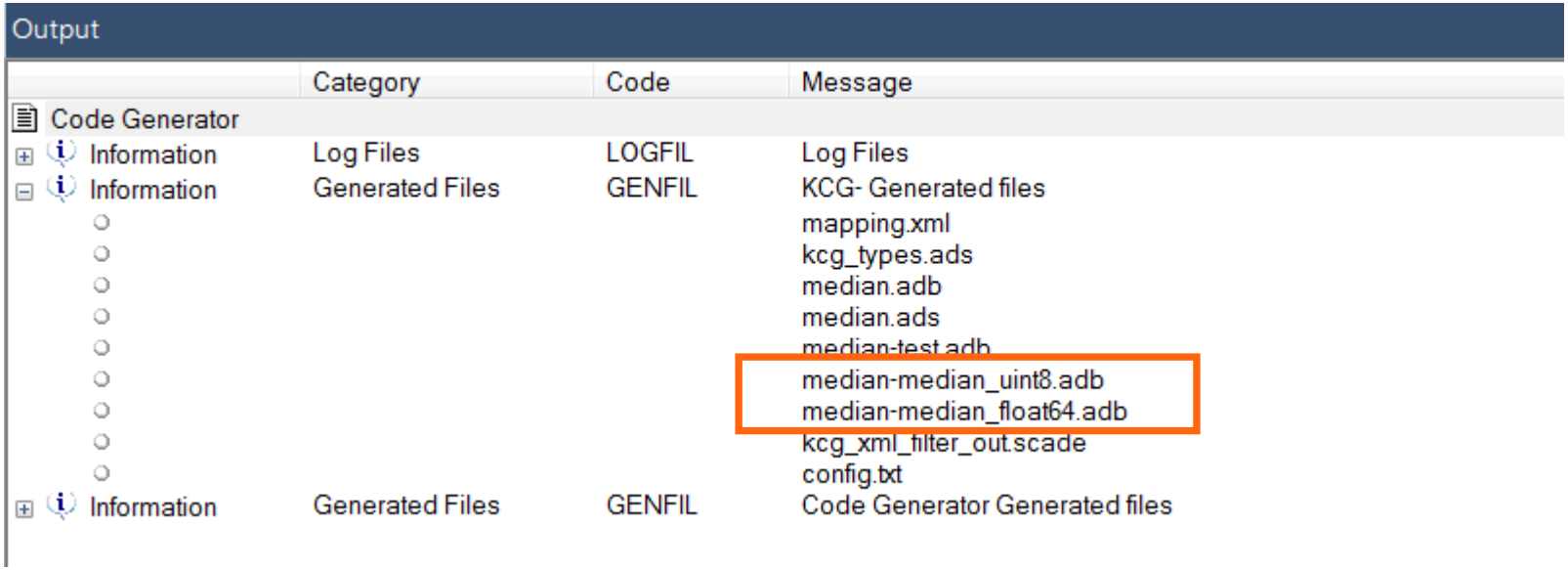
Lab 3: Solution (Ada)

Ada

Lab Solution

Generate the Ada code.

Observe the generated code: for each median operator instance, files are generated.



Lab 4

Lab 4

Lab

Objective:

Constrain to numeric type

Requirements:

Constrain the median input type to be numeric

Time: 5 min

Lab 5

Lab 5: Requirement (2/2)

Lab

Objective:

Prepare the median design environment

Time: 10 min

Requirements:

Modify the `median` implementation to work with arrays of any type and of any size:

- Instantiate it with 3 different array sizes/types

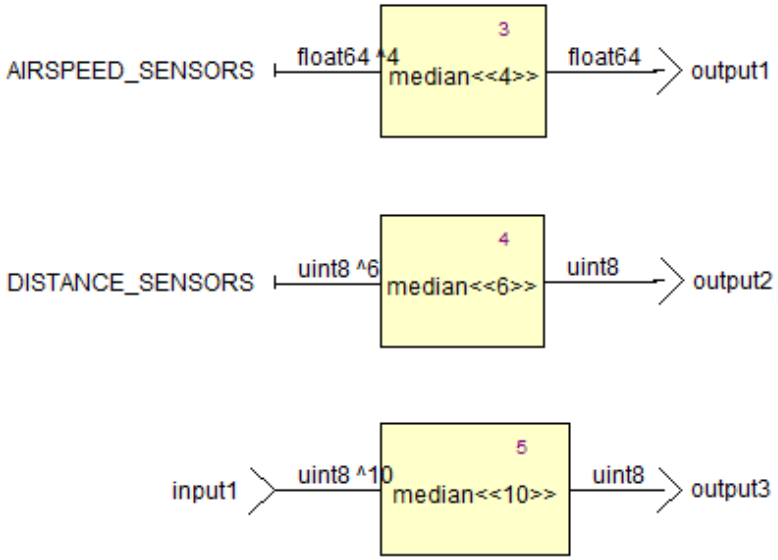
Update `test` operator

- Observe the effects of monomorphisation in the generated code

Lab 5: Requirement (2/2)

Update `test` operator

Name	Kind	Type
input1	input	uint8^10
output1	output	Float64
output2	output	uint8
output3	output	uint8

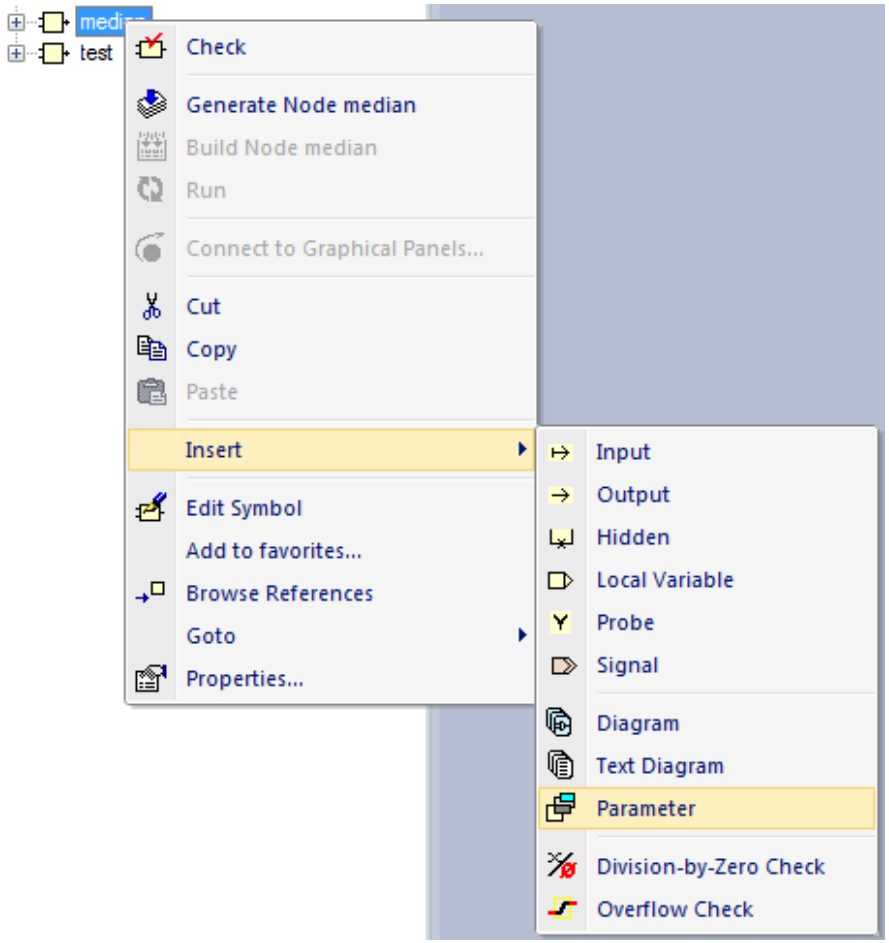


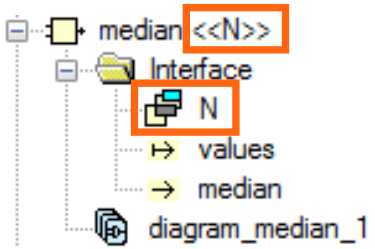
Lab

Lab 5: Prerequisite

Lab Prerequisite

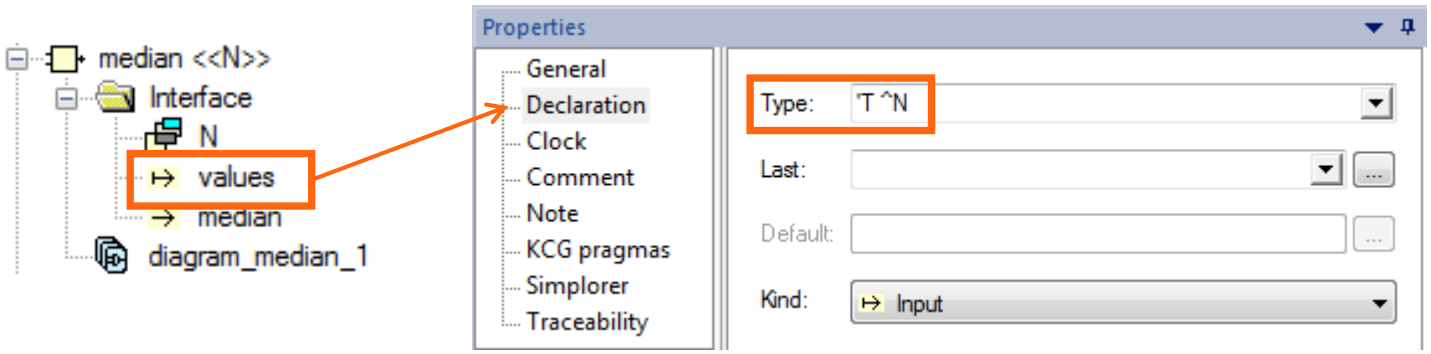
Insert a parameter to have a generic size:



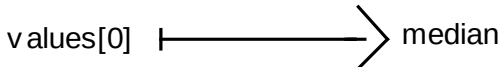


Lab 5: Solution

Change the input type to be generic array, using parameter N as dimension: `T^N



Update the median operator implementation to return values[0]



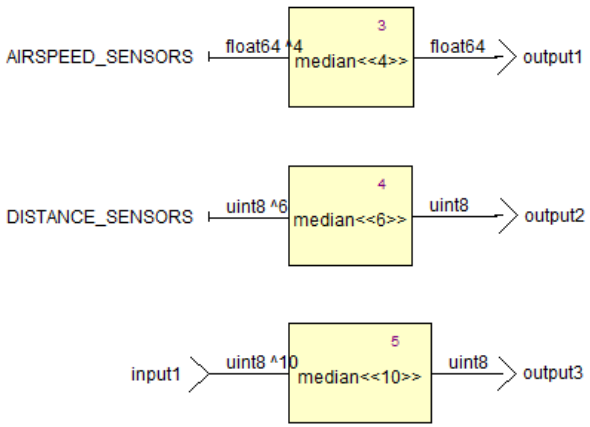
Lab Solution

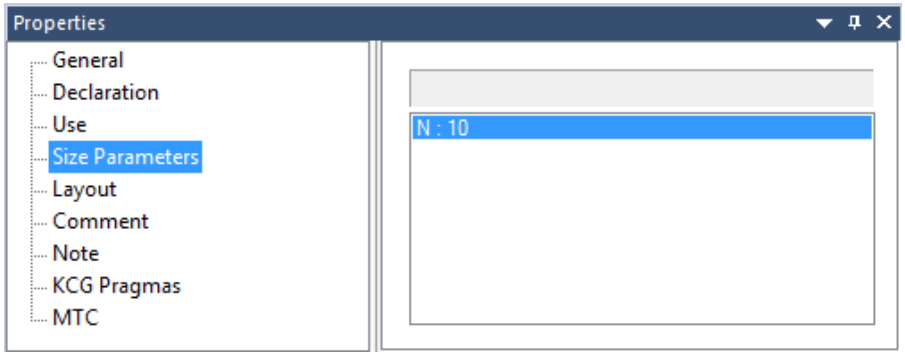
Lab 5: Solution

Lab Solution

Create the “test” operator to use multiple instances of the median operator, with different types and sizes:

- Add one input named input1: uint8^10 and one output, output3:uint8, to test your operator
- Select the operator instances and open Size Parameters Properties to change the parameter value, and use 3 instances of size: 4, 6 and 10
- Create the following design:

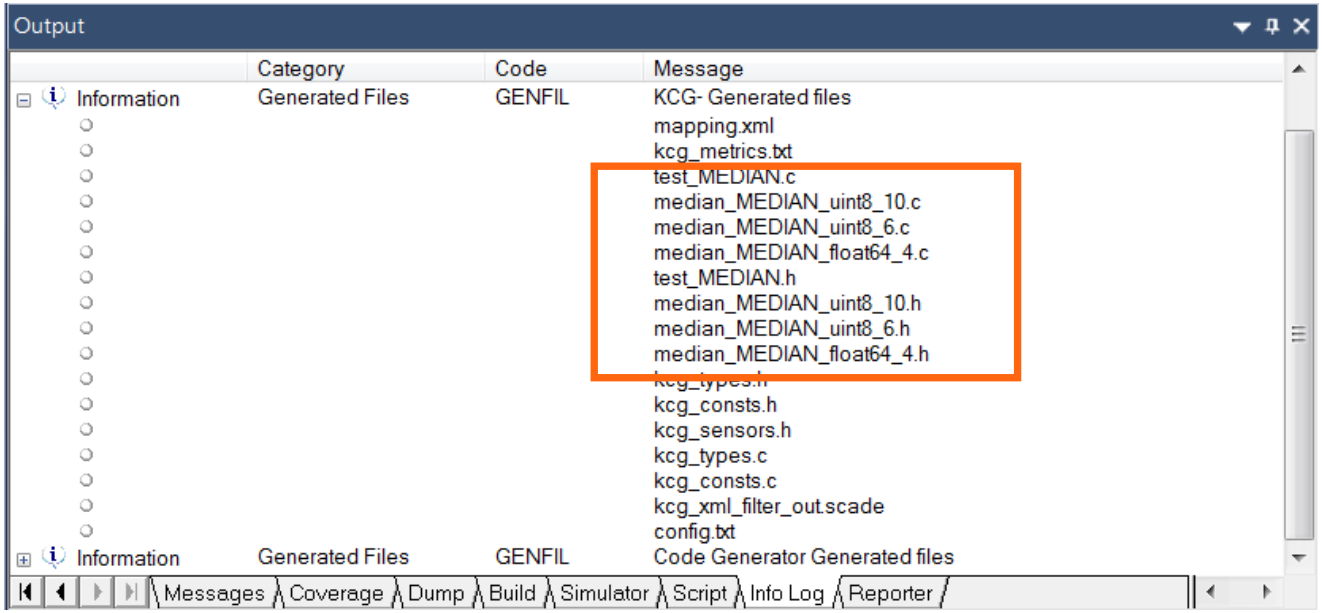




Lab 5: Solution (C Code)

Lab Solution

Generate your C code and observe the generated files:



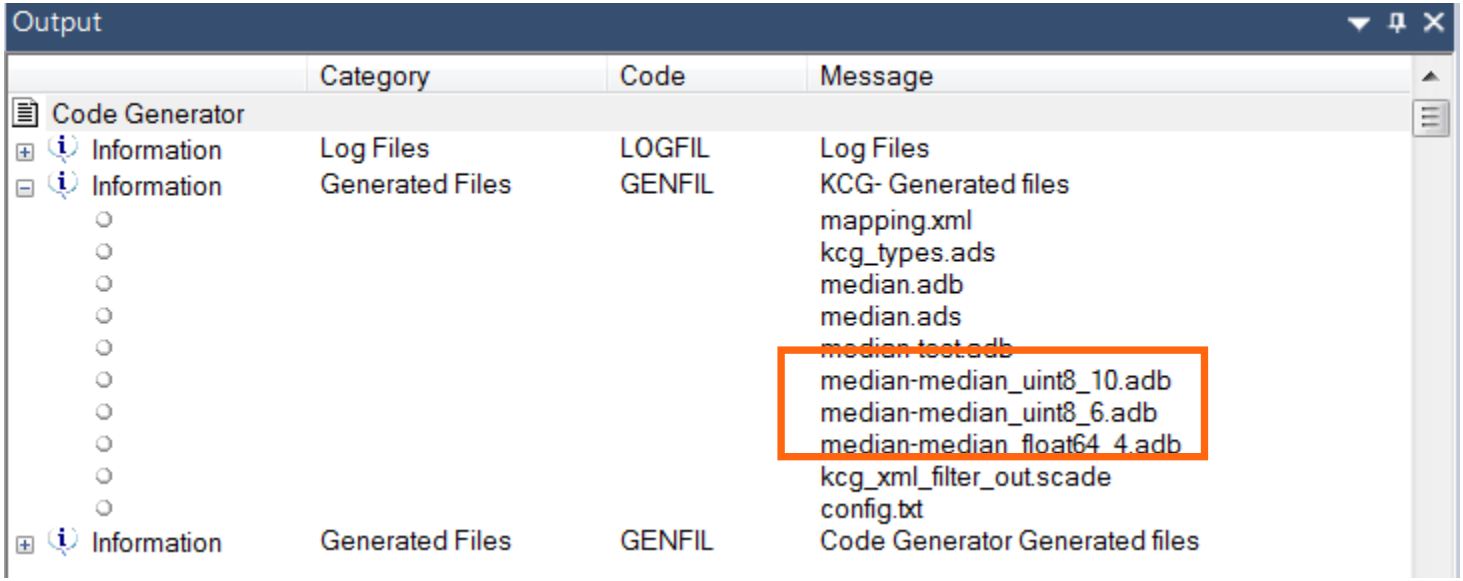
Run and play with the simulator.

Lab 5: Solution (Ada)

Ada

Lab Solution

Generate your Ada code and observe the generated files:



Run and play with the simulator

ANSYS

Lab 6

Lab 6: Requirement

Lab

Objective:

Create a `mean` operator in the median design environment

Time: 10 min

Requirements:

Create a specialized `mean` operator with specialization for integer and floating values

Name	Kind	Type
a	input	'T
b	input	'T
m	output	'T

'T is numeric

Lab 6: Requirement

Create 2 specialized operators:

- `meanInt`

Name	Kind	Type
a	input	uint8
b	input	uint8
m	output	uint8

- `meanReal`

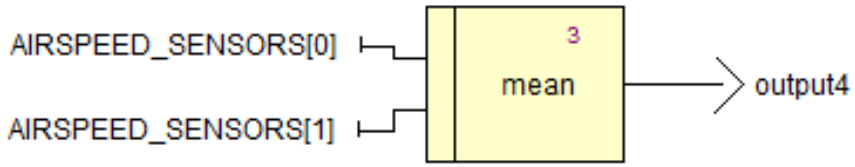
Name	Kind	Type
a	input	float64
b	input	float64
m	output	float64

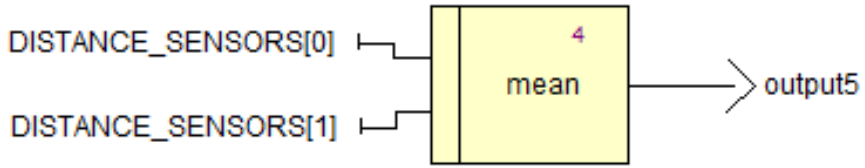
Lab

Lab 6: Requirement

Update `test` operator

Name	Kind	Type
input1	input	uint8^10
output1	output	float64
output2	output	uint8
output3	output	uint8
output4	output	float64
output5	output	uint8





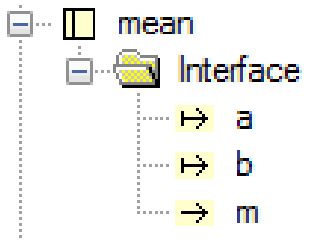
Lab

Lab 6: Solution

Lab Solution

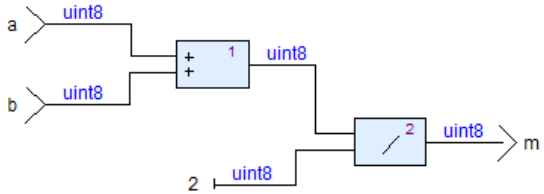
Create the mean imported operator into a new sibling package named Pimp (at the MEDIAN package level), with:

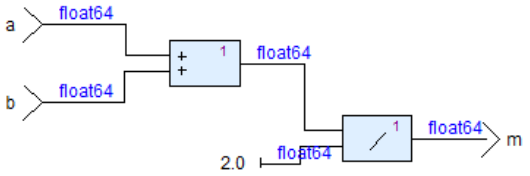
- two inputs a,b:'T
- one output m:'T
- ('T is numeric)



Create two specialization operators into a new sibling package named Pspec (at the MEDIAN package level)

- meanInt(a,b: uint8) return (m: uint8)
- meanReal(a,b: float64) return (m: float64)



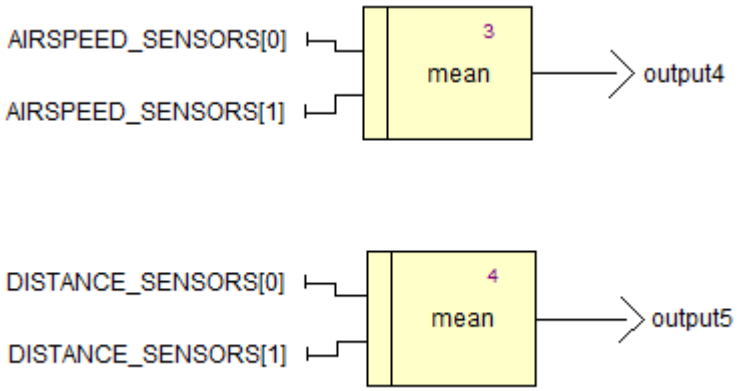


Lab 6: Solution

Lab Solution

Instance the mean in the “test” operator:

- Create new outputs output4 (float) and output5 (integer)
- Use constant values to compute the mean



Run and play with the simulator

Lab 7

Lab 7: Accumulator (1/3)

Lab

Objective:

Create a `min` operator with a mapfold, in the median design environment

Time: 10 min

Requirements:

- Create an operator that provides the minimal numeric value between two inputs.
- Instantiate the `min` operator in the median operator to provide the minimal numeric value of an input vector.

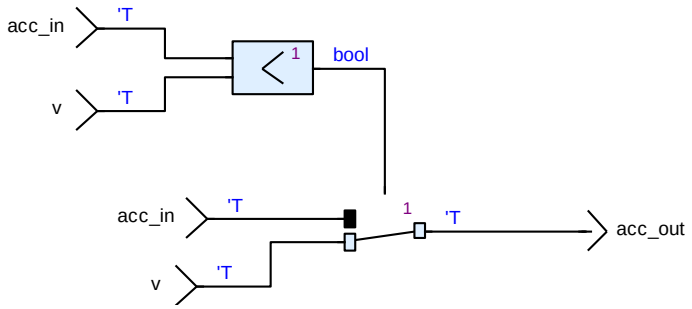
```
min<<N>>(v:'T^N) returns (m:'T)
```

'T is numeric

Lab 7: Accumulator (2/3)

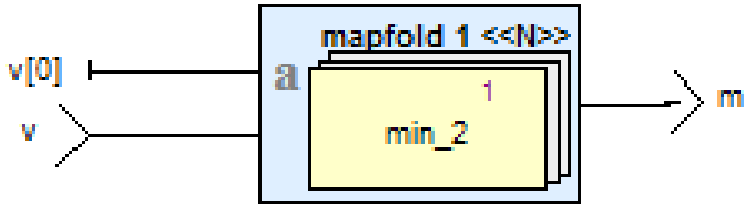
Design a `min_2` operator

Name	Kind	Type
`Acc_in`	input	‘T
`v`	input	‘T
`m`	output	‘T



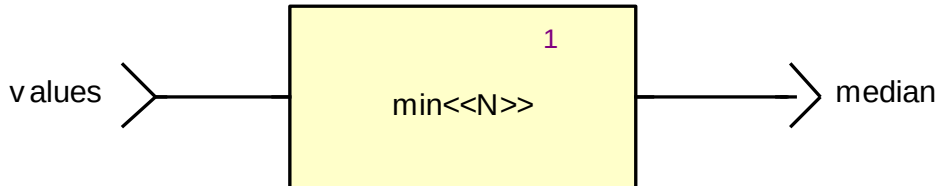
Design the `min` operator

Name	Kind	Type
`v`	input	‘T^N
`m`	output	‘T



Lab 7: Accumulator (3/3)

Udpate the `median` operator

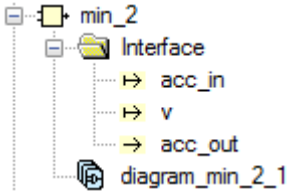


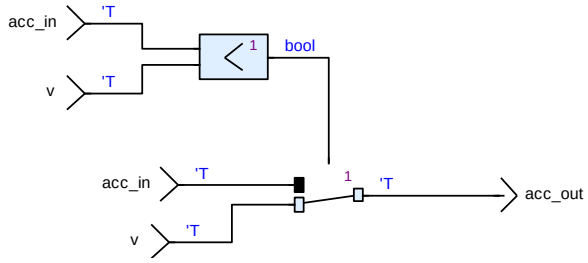
Lab

Lab 7: Solution

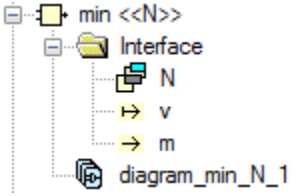
Lab Solution

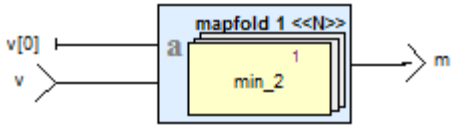
Create a min_2 operator (find minimal value of 2 values):



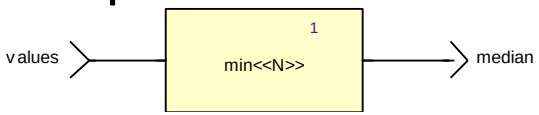


Create a min operator where you instantiate min_2 with a mapfold higher order:





Instantiate the min operator in the median operator



Run and play with simulator on the test operator

Lab 8

Lab 8: Accumulator

Lab

Objective:

Complete the useful operators for the median computation.

Requirements:

Time: 20 min

Create the following operators (use **mapfold**):

Find the maximal numeric value between two values

- `max(acc_in:'T,a:'T) returns (acc:'T)`

Count the number of values greater than the reference value in an input vector

- `Count_gt(acc_in: uint8, ref:'T, a:'T^N) returns (acc:uint8)`

Count the number of values lower than the reference value in an input vector

- `Count_lt(acc_in:uint8, ref:'T, a:'T^N) returns (acc:uint8)`

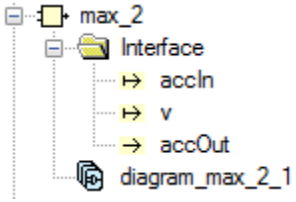
'T is numeric

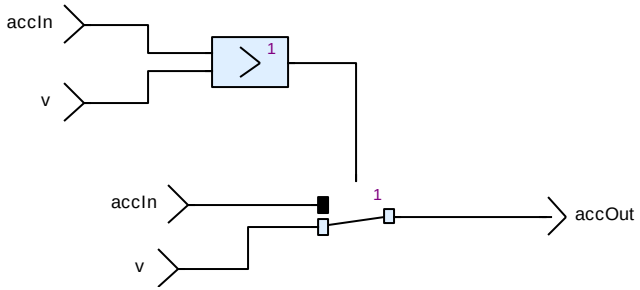
ANSYS

Lab 8: Solution

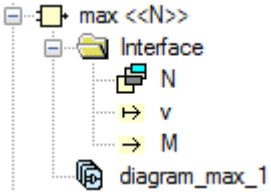
Lab Solution

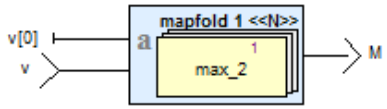
Create a max_2 operator (find maximal value of 2 values):





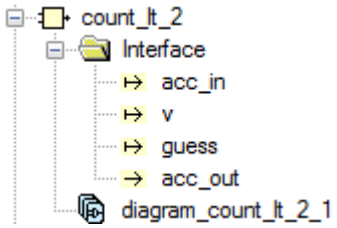
Create a max operator where you instantiate max_2 with mapfold higher order:

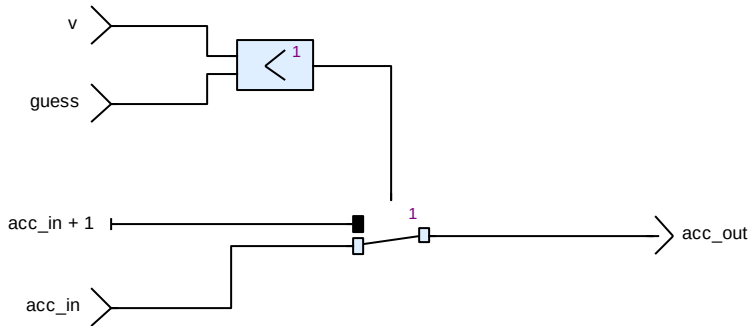




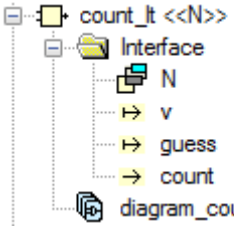
Lab 8: Solution

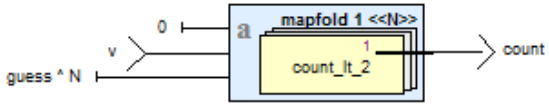
Create a count_lt_2 operator:





Create a count_lt operator where you instantiate count_lt_2 with mapfold higher order:





Repeat the steps for count_gt_2 and count_gt

Lab 9

Lab 9: Torben Algorithm

Lab

Objective:

Complete the median design

Time: 20 min

Requirements:

Use the Torben algorithm to complete the design of the median

Lab 9: Prerequisite

Lab
Prerequisite

http://ndevilla.free.fr/median/median/index.html

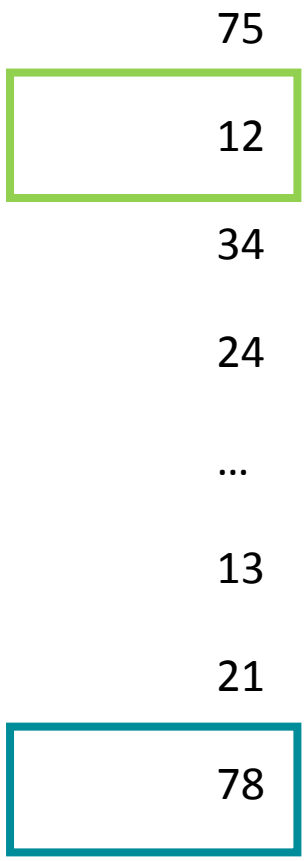
Median filtering is a commonly used technique in signal processing. Typically used on signals that may contain outliers skewing the usual statistical estimators, it is usually considered too expensive to be implemented in real-time or CPU-intensive applications.

In a safety environment, the worst case execution time (WCET) is more important than the best execution time. Also, we prefer to avoid the large memory copy even if the algorithm is slower.

ANSYS

Torben's Algorithm

1. Start by finding max and min values
2. And make a guess : the median should be the mean between this min and max value : $\frac{min+max}{2}$
3. min = 12, max = 78, guess = 45



Lab Prerequisite

Torben's Algorithm

1. min and max represent the minimal and maximal value of the algorithm.
2. We now count the number of values that are lower than our guess ant the number that are above our guess :
   5 are smaller than 45, 2 are above
3. This means that our median is between our guess and the previous min value: we can use our guess as the new max value. We can even improve this by using the value immediately below our guess as the new max (as our max is possibly not present in our lis)
4. The value immediately below 45 is 34, this becomes our new max

Lab Prerequisite

Torben's Algorithm

1. Our new guess is then (12+34)/2 = **23**
2. below = 3, above = 4
3. The definition of the median is to have as many above than below. So we know in this case that our median is the value immediately above our guess, that is **24**
4. This is an n logn algorithm

Lab Prerequisite

Lab 9: Prerequisite (C Code)

Lab Prerequisite

```c
/*
 * The following code is public domain.
 * Algorithm by Torben Mogensen, implementation by N. Devillard.
 * This code in public domain.
 */

#include "median_c.h"

elem_type torben(elem_type *m, size_array n)
{
    int         i, less, greater, equal;
    elem_type  min, max, guess, maxltguess, mingtguess;

    min = max = m[0] ;
    for (i=1 ; i<n ; i++) {
        if (m[i]<min) min=m[i];
        if (m[i]>max) max=m[i];
    }

    while (1) {
        guess = (min+max)/2;
        less = 0; greater = 0; equal = 0;
        maxltguess = min ;
        mingtguess = max ;
        for (i=0; i<n; i++) {
            if (m[i]<guess) {
                less++;
                if (m[i]>maxltguess) maxltguess = m[i] ;
            } else if (m[i]>guess) {
                greater++;
                if (m[i]<mingtguess) mingtguess = m[i] ;
            } else equal++;
        }
        if (less <= (n+1)/2 && greater <= (n+1)/2) break ;
        else if (less>greater) max = maxltguess ;
        else min = mingtguess;
    }
    if (less >= (n+1)/2) return maxltguess;
    else if (less+equal >= (n+1)/2) return guess;
    else return mingtguess;
}
```

min and max initialization

guess computation

counting the distribution around guess value, select new min and max

stop condition

final median value

Lab 9: Prerequisite (C Code)

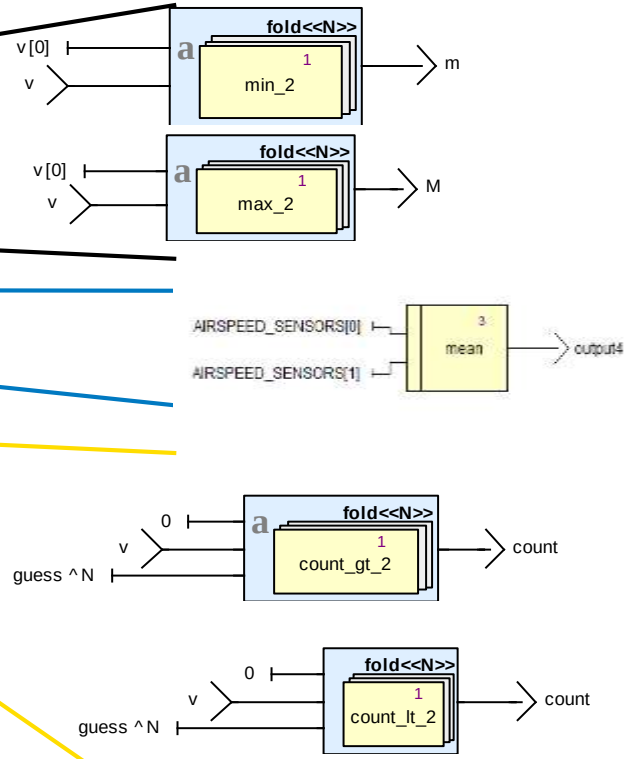
Lab Prerequisite

C code algorithm

```
 * The following code is public domain.
 * Algorithm by Torben Mogensen, implementation by N. Devillard.
 * This code in public domain.
 */

#include "median_c.h"

elem_type torben(elem_type *m, size_array n)
{
    int         i, less, greater, equal;
    elem_type  min, max, guess, maxltguess, mingtguess;

    min = max = m[0] ;
    for (i=1 ; i<n ; i++) {
        if (m[i]<min) min=m[i];
        if (m[i]>max) max=m[i];
    }

    while (1) {
        guess = (min+max)/2;
        less = 0; greater = 0; equal = 0;
        maxltguess = min ;
        mingtguess = max ;
        for (i=0; i<n; i++) {
            if (m[i]<guess) {
                less++;
                if (m[i]>maxltguess) maxltguess = m[i] ;
            } else if (m[i]>guess) {
                greater++;
                if (m[i]<mingtguess) mingtguess = m[i] ;
            } else equal++;
        }
        if (less <= (n+1)/2 && greater <= (n+1)/2) break ;
        else if (less>greater) max = maxltguess ;
        else min = mingtguess;
    }
    if (less >= (n+1)/2) return maxltguess;
    else if (less+equal >= (n+1)/2) return guess;
    else return mingtguess;
}
```

SCADE Suite model already designed

Use mapfold



Lab 9: Prerequisite (Ada)

Ada

```
function torben(m:Kcg_Types.Array_Float64; n:Integer) return Kcg_Config.Kcg_Float64
is
less, greater, equal: Integer;
min, max : Kcg_Config.Kcg_Float64:=m(0);
guess, maxltguess, mingtguess : Kcg_Config.Kcg_Float64;
flag : Boolean :=true;

begin
  for i in 1..n loop
        if m(i) < min then
                min := m(i);
        end if;
        if m(i) > max then
                max := m(i);
        end if;
  end loop;
        while (flag) loop
                guess := (min + max)/2.0;
                less:= 0; greater:= 0; equal:= 0;
                maxltguess := min;
                mingtguess := max;
                for i in 0..n loop
                        if m(i) < guess then
                                less := less + 1;
                                if m(i) > maxltguess then
                                        maxltguess := m(i);
                                end if;
                        end if;
                        if m(i) > guess then
                                greater := greater + 1;
                                if m(i)< mingtguess then
                                        mingtguess := m(i);
                                end if;
                        end if;
                        if m(i) = guess then
                                equal:= equal + 1;
                        end if;
                end loop;
                if (less <= (n+1)/2 and greater <= (n+1)/2) then flag :=false;
                end if;
                if (less>greater) then
                        max := maxltguess;
                else
                        min := mingtguess;
                end if;
        end loop;
        if (less >= (n+1)/2) then return maxltguess;
        end if;
        if (less+equal >= (n+1)/2) then return guess;
        else return mingtguess;
        end if;
end torben;
```

min and max initialization

guess computation

counting the distribution around guess value, select new min and max

stop condition

final median value

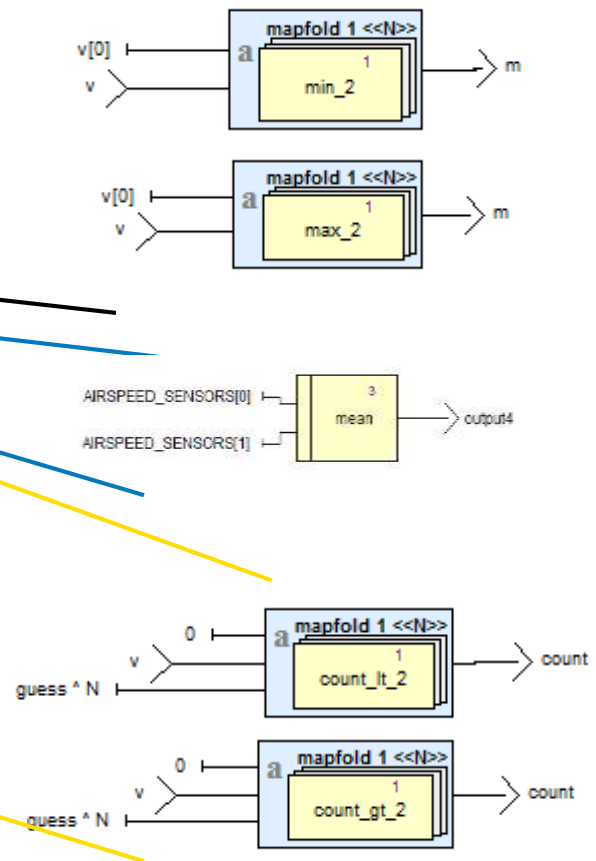
ANSYS

Lab 9: Prerequisite (Ada)

Ada

```
function torben(m:Kcg_Types.Array_Float64; n:Integer) return Kcg_Config.Kcg_Float64
is
less, greater, equal: Integer;
min, max : Kcg_Config.Kcg_Float64:=m(0);
guess, maxltguess, mingtguess : Kcg_Config.Kcg_Float64;
flag : Boolean :=true;

begin
 for i in 1..n loop
       if m(i) < min then
              min := m(i);
       end if;
       if m(i) > max then
              max := m(i);
       end if;
  end loop;
       while (flag) loop
              guess := (min + max)/2.0;
              less := 0; greater := 0; equal := 0;
              maxltguess := min;
              mingtguess := max;
              for i in 0..n loop
                     if m(i) < guess then
                            less := less + 1;
                            if m(i) > maxltguess then
                                   maxltguess := m(i);
                            end if;
                     end if;
                     if m(i) > guess then
                            greater := greater + 1;
                            if m(i)< mingtguess then
                                   mingtguess := m(i);
                            end if;
                     end if;
                     if m(i) = guess then
                            equal:= equal + 1;
                     end if;
              end loop;
              if (less <= (n+1)/2 and greater <= (n+1)/2) then flag :=false;
              end if;
              if (less>greater) then
                     max := maxltguess;
              else
                     min := mingtguess;
              end if;
       end loop;
       if (less >= (n+1)/2) then return maxltguess;
       end if;
       if (less+equal >= (n+1)/2) then return guess;
       else return mingtguess;
       end if;
end torben;
```

SCADE Suite model already designed



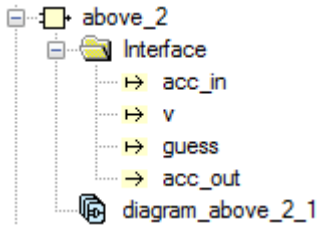
Lab 9: Solution

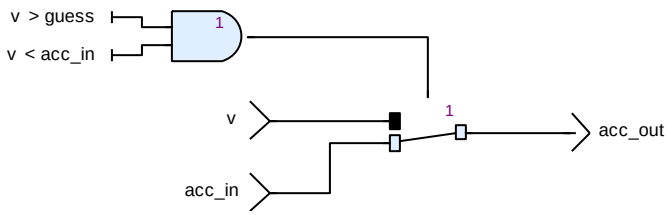
Lab Solution

You need to implement a new min/max selection and the median value finalization

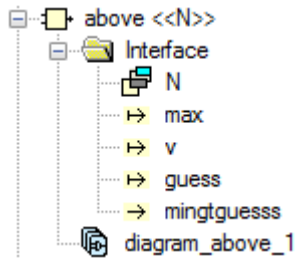
1- Implement “above” operator to define new min/max value for next iteration:

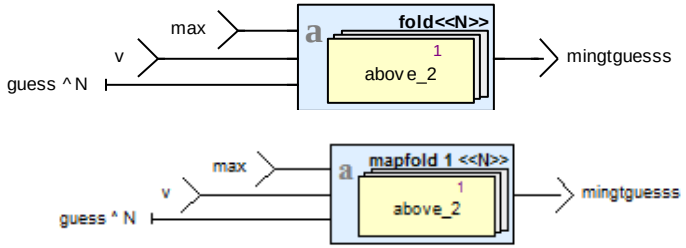
above_2(accIn,v,guess:’T) returns (accOut:’T)





above<<N>>(max:’T,v:’T^N,guess:’T) returns (mingtguess:’T)



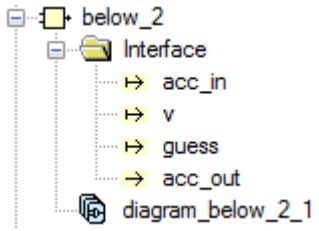


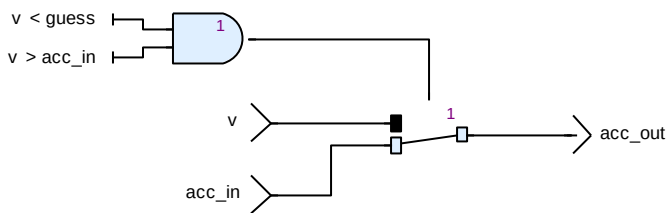
Lab 9: Solution

Lab Solution

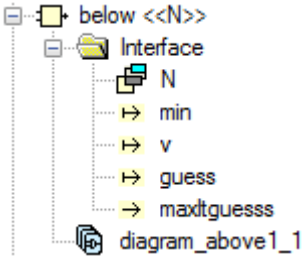
2- Implement the “below” operator to define new min/max value for next iteration:

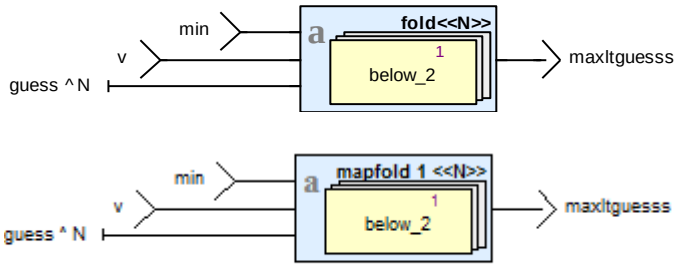
below_2(accIn,v,guess:’T) returns (accOut:’T)





below<<N>>(min:’T,v:’T^N,guess:’T) returns (maxltguess:’T)



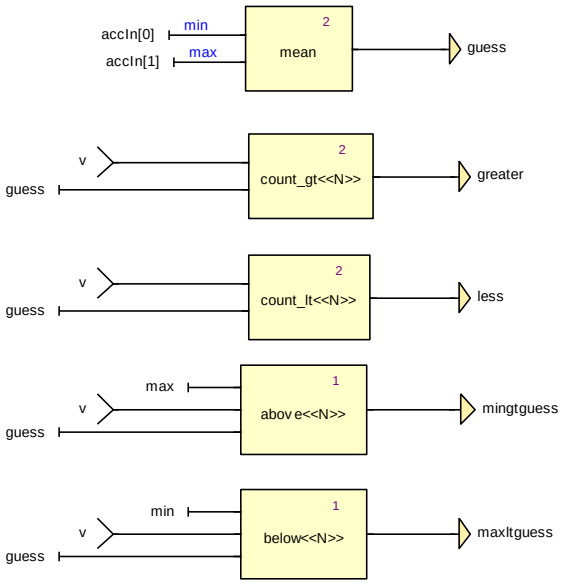


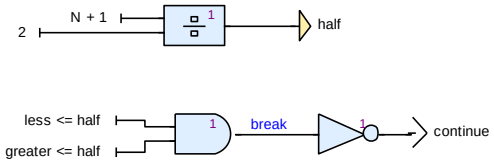
Lab 9: Solution

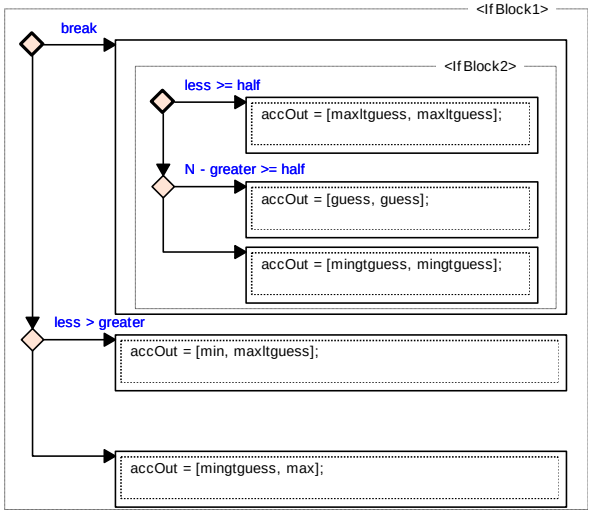
3- Create “guess<<N>>” operator to compute the median.

- This operator is mapfoldw to find the median value
- The accumulator is min and max value, stored in array

guess<<N>>(accIn:’T^2,v :’T^N) returns (continue:bool,accOut:’T^2)





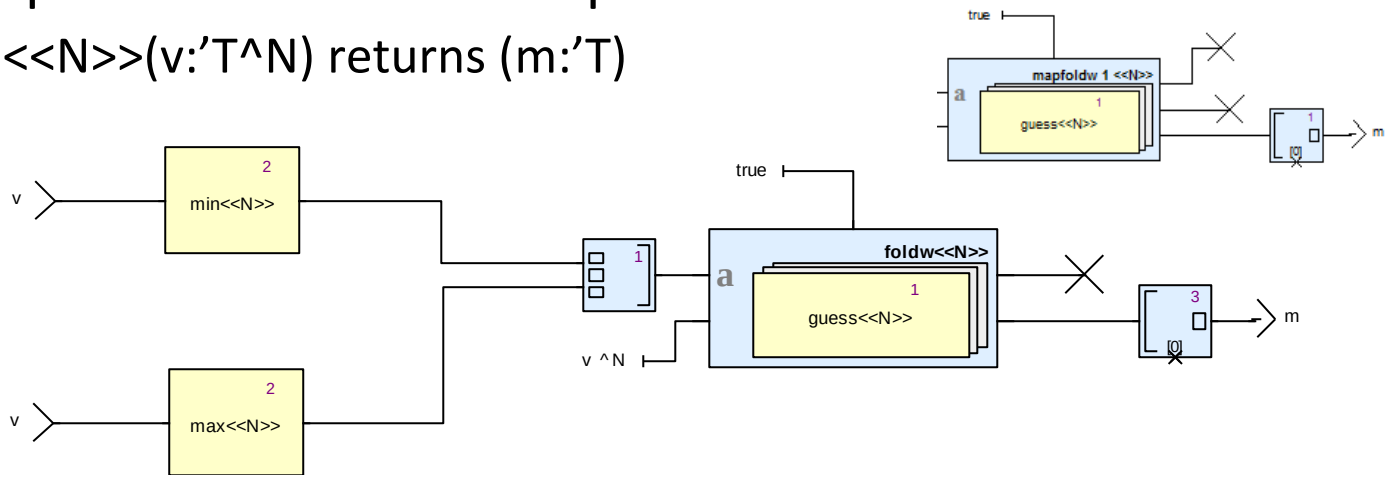


Lab 9: Solution

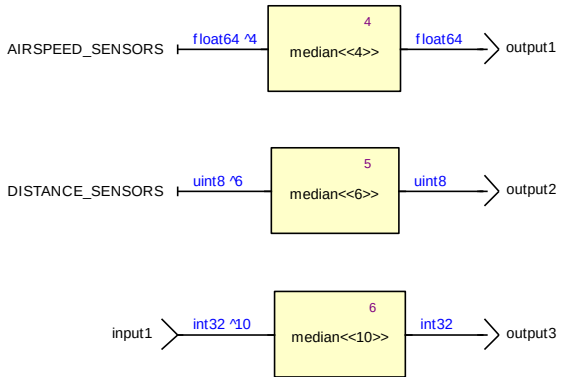
Lab Solution

4- Complete median<<N>> operator:

median<<N>>(v:'T^N) returns (m:'T)



5- Test median<<N>> operator (you can use the current created environment):



Lab 10

Lab 10: KCG

Lab

Objective:

Understand the generated files

Time: 10 min

Requirements:

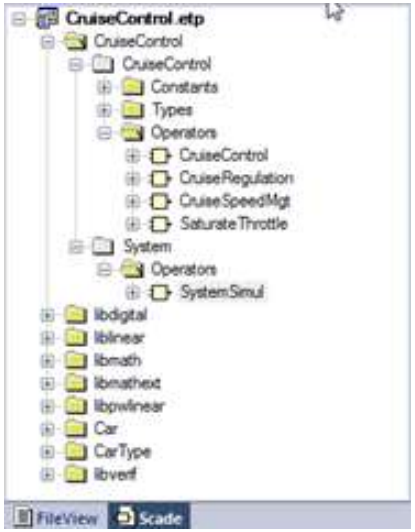
Generate KCG code of your median design, with default options

Observe the generated files (in KCG or KCGAda directory)

Lab 10: Generate Code from IDE

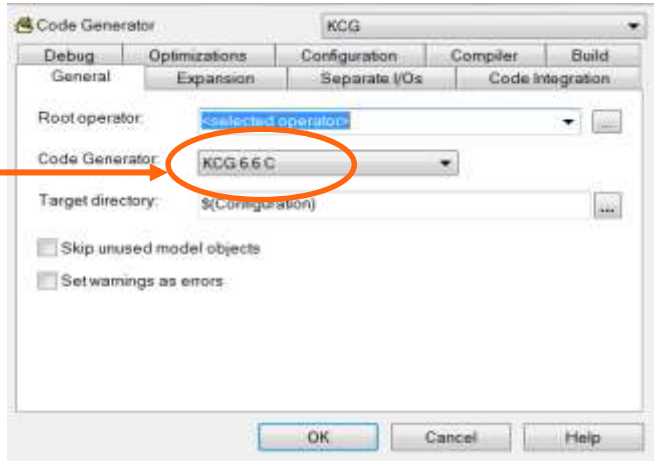
Lab Prerequisite

Select the root operator from Scade view



Click on Settings in the Code Generator toolbar:

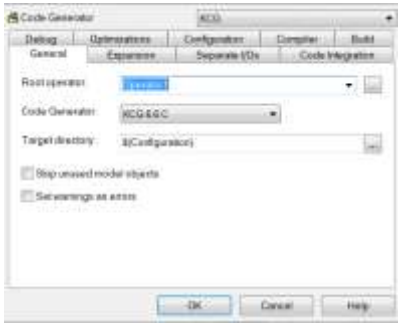
- Select the Code Generator in the General tab:
  - KCG 6.6 C or KCG 6.6 Ada



Lab 10: Generate Code From IDE

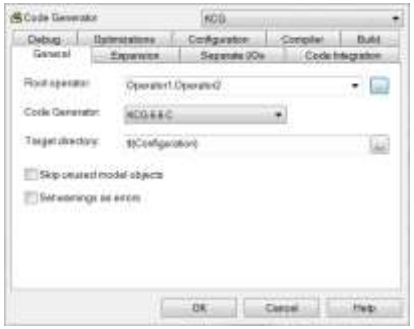
Lab Prerequisite

You can also specify the root operator from the “Root node” drop-down list



It is possible to set several root operators:

- Click on the browsing button and check / uncheck nodes:
- All selected operators are displayed separated by a comma in the Root node field



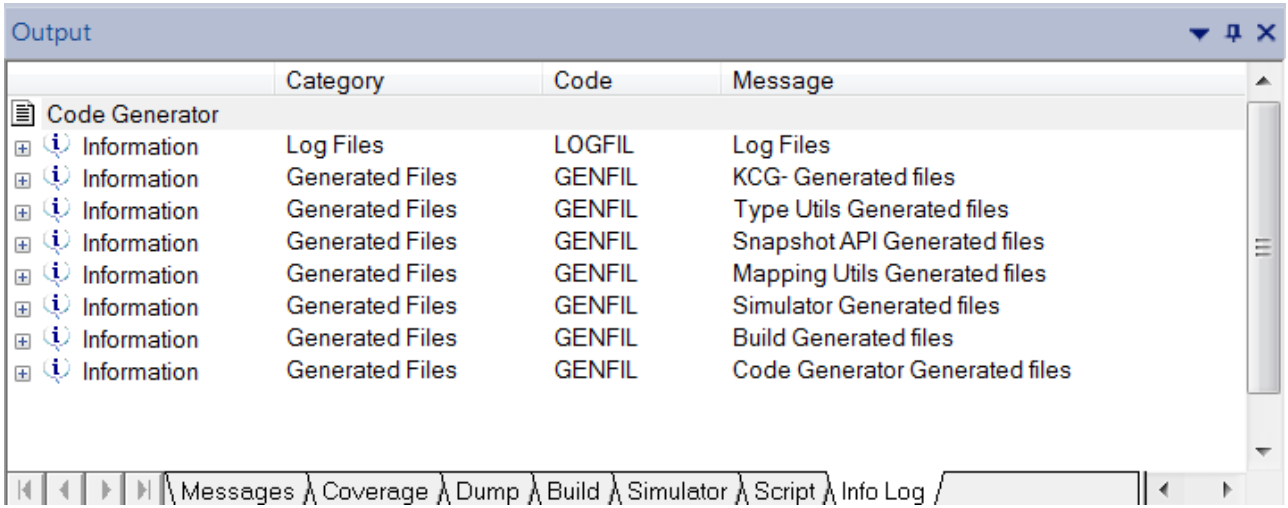
Lab 10: Generate Code From IDE

Lab Prerequisite

Generate the code based on the active configuration set in the Code Generator toolbar:

Check in the “Info Log” tab:

- If the generation code is successful,
- If errors messages are displayed,
- The list of generated files.



Lab 10: Generate Qualified / Certified Code

KCG is qualified / certified in a specific usage context (batch mode), described in its requirements document or SCADE Technical document.

Kcg<version>.exe <kcg options> <SCADE model files>

Example:

- Generate C code
  - kcg66.exe –root <NodeName> -expall -target C MySCADEModel.xscade MySCADEModel2.xscade
- Generate Ada code
  - kcg66.exe –root <NodeName> -expall -target Ada MySCADEModel.xscade MySCADEModel2.xscade

Lab Prerequisite

Lab 11

Lab 11:

Objective:

Understand KCG options

Requirements:

Generate KCG code of your median design, with different options:

- Change the target directory
- Modify the expansion options
- Generate the context as global (C code)
- Change the name length and significance length (C code)
- Change the top-level package (Ada code)

…

For each generation, observe the generated files (in KCG or KCGAda directory)

Lab

Lab 12

Lab 12: Imported Operator

Lab

Objective:

Import target code (C or Ada) and simulate it

Requirements:

Time: 15 min

Use torben () function (median_c.c or median_a.adb file) to test your design

Add a test with the imported function and simulate

Lab 12: C Wrapping Function

Lab Solution

You need to encapsulate the torben function to use in your SCADE design

In Prerequisites\median_c.c file, below torben definition function, create the function:

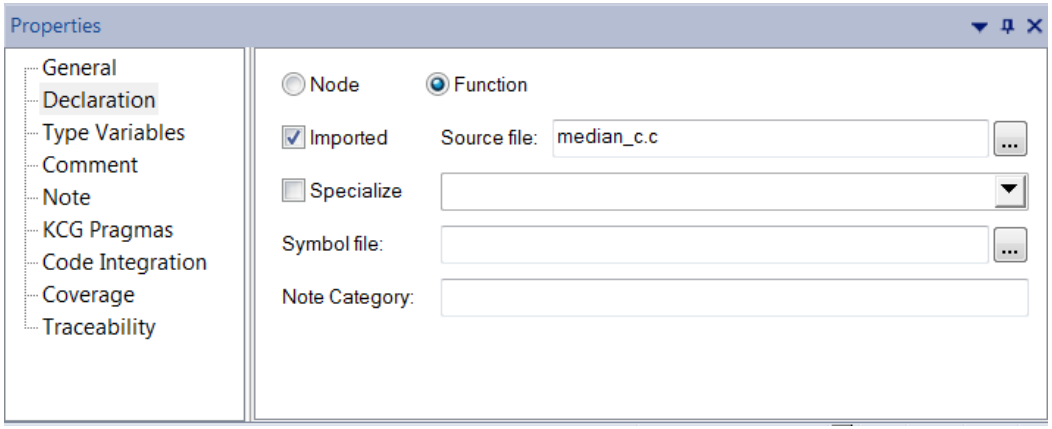
```
kcg_float64 torben_real_4_Up(kcg_float64 m[4])
{
   elem_type array[4];
   int i;
   for (i=0; i<4; i++)
     array[i] = m[i];
   return (kcg_float64) torben(array, 4);
}
```

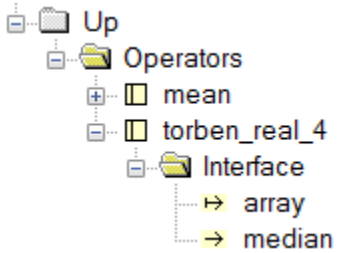
Lab 12: Imported SCADE Function (C Code)

Lab Solution

Create the imported function (into Up package)

- torben_real_4(array: float64^4) returns (median: float64)





Lab 12: Ada Wrapping Function

Ada

Lab Solution

You need to encapsulate the torben function to use in your SCADE design:

In Prerequisites\Up.adb file, below torben definition function, create the function:

```
-- Pimp::torben_real_4/
function torben_real_4(
  -- array/
   array_1 : in Kcg_Types.Float64_Range_0_3) return Kcg_Config.Kcg_Float64
is

  median_2 : Kcg_Config.Kcg_Float64;

begin
  median_2 := torben(m=>array_1,n=>3);
return median_2;

end torben_real_4;
```

ANSYS

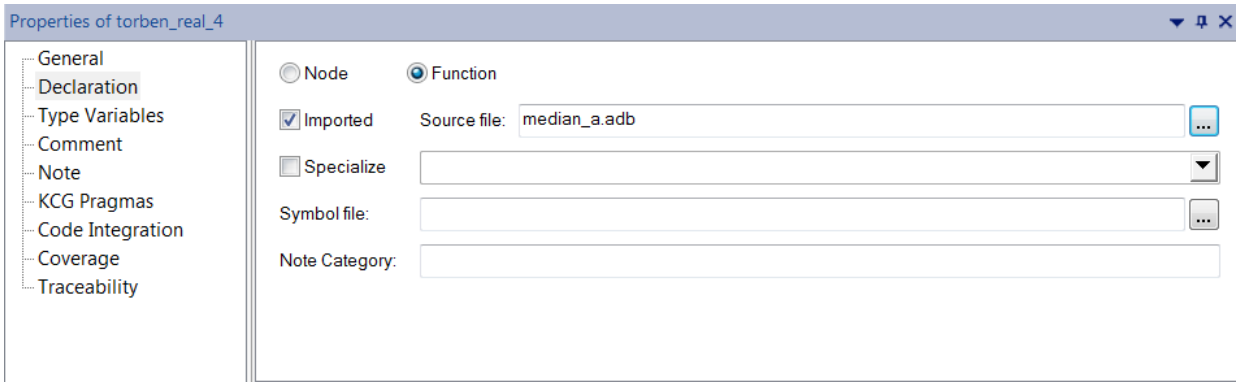
Lab 12: Imported SCADE Function (Ada)

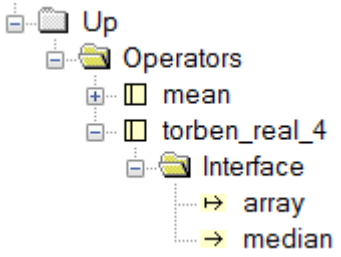
Ada

Lab Solution

Create the imported function into Up package:

- torben_real_4(array: float64^4) returns (median : float64)

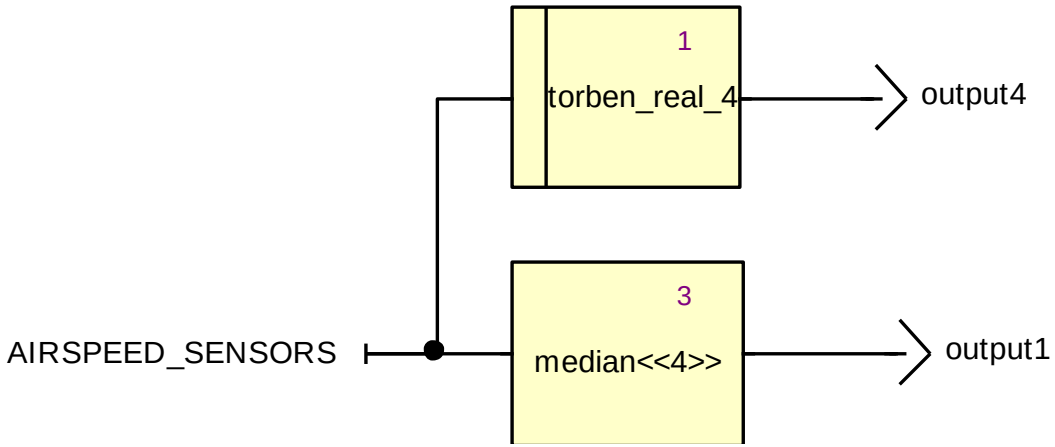




ANSYS

Lab 12: Imported SCADE Function

Test torben_real_4 imported operator in “test” operator:



Lab Solution

ANSYS TechTips

ANSYS Techtips

Find the complete design in video on ANSYS Techtips Youtube channel

https://www.youtube.com/playlist?list=PL0lZXwHtV6Ombl6N3xguI86cpQ44GxTuX

Contacts

Legal Contact
Esterel Technologies SAS
14/15, Place Georges Pompidou
78180 Montigny Le Bretonneux
FRANCE
Phone: +33 1 30 68 61 60
Fax: +33 1 30 68 61 61

Technical Support
Esterel Technologies SAS
Parc Avenue - 9 rue Michel Labrousse
31100 Toulouse FRANCE
Phone: +33 5 34 60 90 50
Fax: +33 5 34 60 90 41

Submit questions to Technical Support: scade-support@ansys.com
Contact one of our Sales representatives at: scade-sales@ansys.com
Direct general questions about Esterel Technologies to: scade-info@ansys.com
Discover the latest news on our products and technology at: http://www.ansys.com/products/embedded-software

Legal Information