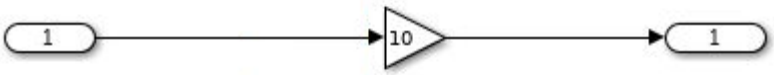
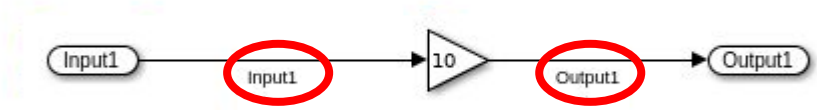
## *Creazione del modello*

Per essere compatibile con la GAM il modello deve ricevere gli ingressi da blocchi “Inport” e mandare le uscite su blocchi “Outport”.
Creare un modello vuoto e inserire blocchi e collegamenti come in figura:




Perché il segnale del sia correttamente mappato dalla GAM ogni segnale di ingresso e uscita deve avere un nome: doppio click sulla linea che rappresenta il segnale per dare un nome a quel segnale:




## *Configurazione dei parametri*

I parametri dei blocchi possono essere statici (non più modificabili dopo la generazione del codice) oppure tunable (modificabili in runtime).
Per avere parametri tunable impostare i parametri come variabile:


Il rosso indica che il parametro non è definito nel workspace corrente. Definirlo dalla console:

```
>> paramVar1 = 10

paramVar1 =

    10
```

Il modello si presenta così:


Ora è richiesto di configurare il modello per usare i parametri tunable:

Model Setting > Code Generation > Optimization, impostare “Default parameter behaviour” su “Tunable”
Questo step non è richiesto se poi viene usato il programma riportato più avanti (che si occupa anche di settare questa impostazione)

## *Configurazione del modello per la generazione del codice*

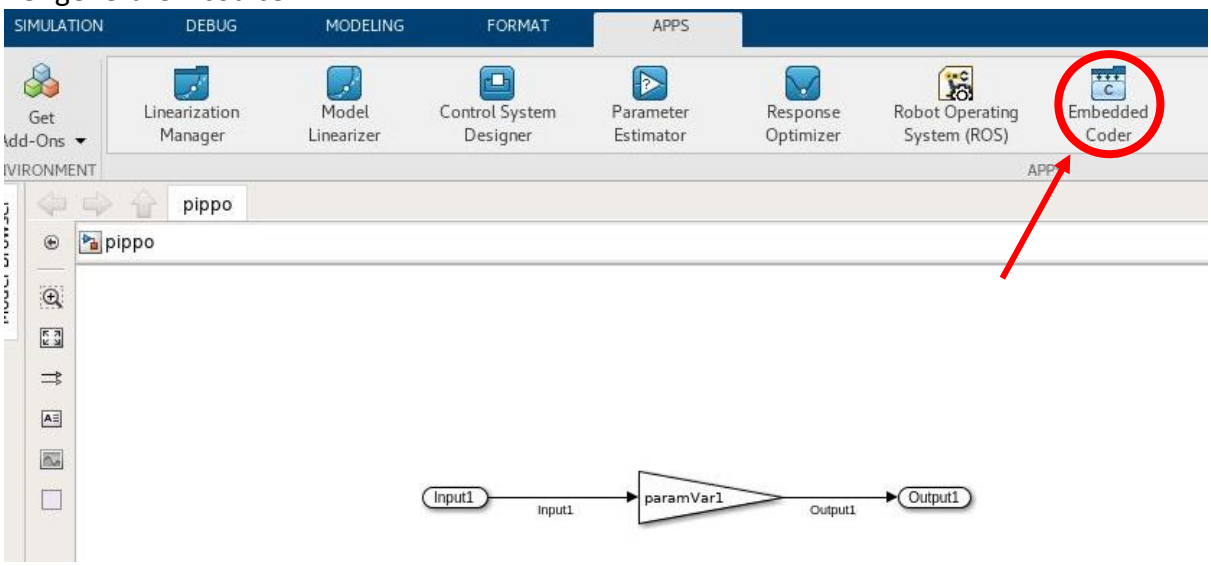
Per configurare il modello per la generazione di codice copiare e lanciare questo programma (dopo aver modificato la variabile model_name con il nome del modello corrente):

```
model_name = “nome_del_modello”;

% Solver
set_param(model_name, 'SolverType', 'Fixed-step');

% Code Generation
set_param(model_name, 'SystemTargetFile', 'ert_shrlib.tlc');
set_param(model_name, 'RTWVerbose',       0              );

% Optimization
set_param(model_name, 'DefaultParameterBehavior', 'Tunable');
set_param(model_name, 'OptimizationCustomize',    1        );
set_param(model_name, 'GlobalVariableUsage',      'None'   );

% Report
set_param(model_name, 'GenerateReport', 0);

% Comments
set_param(model_name, 'GenerateComments', 0);

% Custom code (MODEL is a coder varialbe for the model name)
set_param(model_name, 'CustomSourceCode', ...
    [ ...
    '#define CONCAT(str1, str2, str3) CONCAT_(str1, str2, str3)'            newline, ...
    '#define CONCAT_(str1, str2, str3) str1 ## str2 ## str3'                newline, ...
    '#define GET_MMI_FUNC    CONCAT(MODEL     , _GetCAPImmi ,    )'         newline, ...
    '#define RT_MODEL_STRUCT CONCAT(RT_MODEL_ , MODEL       , _T )'         newline, ...
                                                                            newline, ...
    'void* GET_MMI_FUNC(void* voidPtrToRealTimeStructure)'                  newline, ...
    '{'                                                                     newline, ...
    '   rtwCAPI_ModelMappingInfo* mmiPtr = &(rtmGetDataMapInfo( ( RT_MODEL_STRUCT *
)(voidPtrToRealTimeStructure) ).mmi);' newline, ...
    '   return (void*) mmiPtr;'                                             newline, ...
    '}' ...
    ] ...
);

% Interface
set_param(model_name, 'SupportComplex',      0);
set_param(model_name, 'SupportAbsoluteTime', 0);
set_param(model_name, 'SuppressErrorStatus', 1);

set_param(model_name, 'CodeInterfacePackaging', 'Reusable function');

set_param(model_name, 'RootIOFormat', 'Part of model data structure');

set_param(model_name, 'RTWCAPIParams', 1);
set_param(model_name, 'RTWCAPIRootIO', 1);

set_param(model_name, 'GenerateAllocFcn', 1);

set_param(model_name, 'IncludeMdlTerminateFcn',    0);
set_param(model_name, 'CombineSignalStateStructs', 1);

set_param(model_name, 'ArrayLayout', 'Column-major');

% Templates
set_param(model_name, 'GenerateSampleERTMain', 0);
```

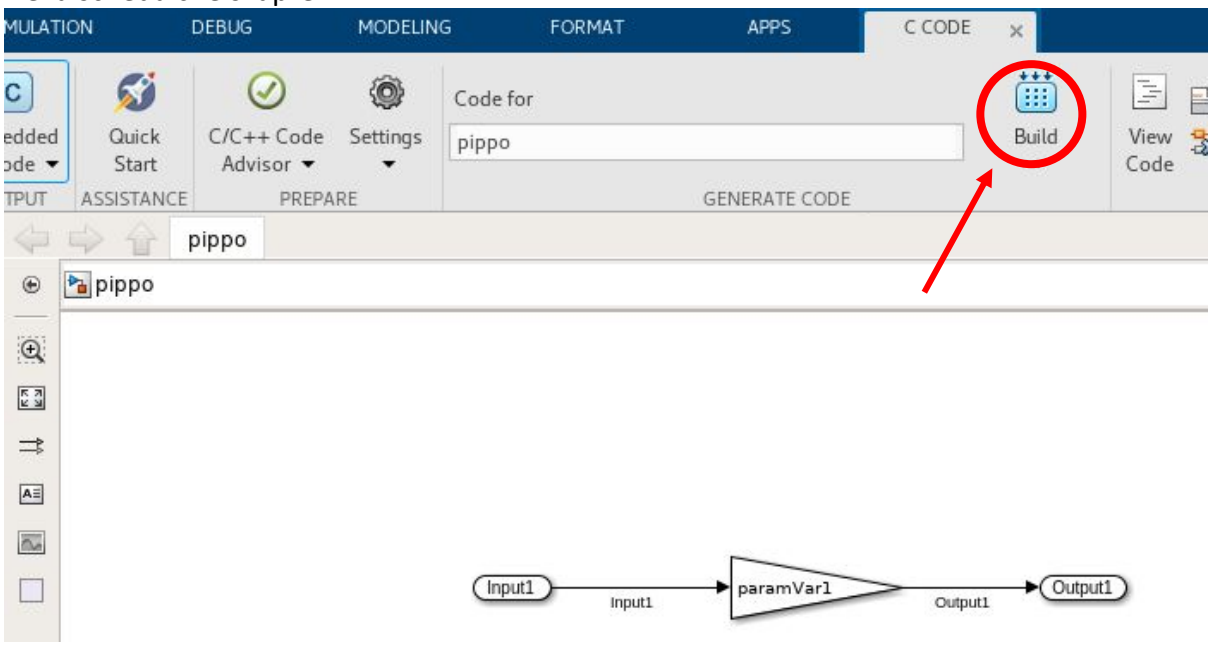
Il programma configura correttamente il modello per la generazione (imposta i settaggi richiesti in Model Settings).

## *Generazione del codice*

Per generare il codice:




Nella scheda che si apre:




Confermare con OK eventuali altre richieste del modello.
Nella cartella di lavoro corrente si ottiene un file nome_modello.so che è il codice generato dal modello.

## *Uso del modello nella GAM*

Installazione di marte:

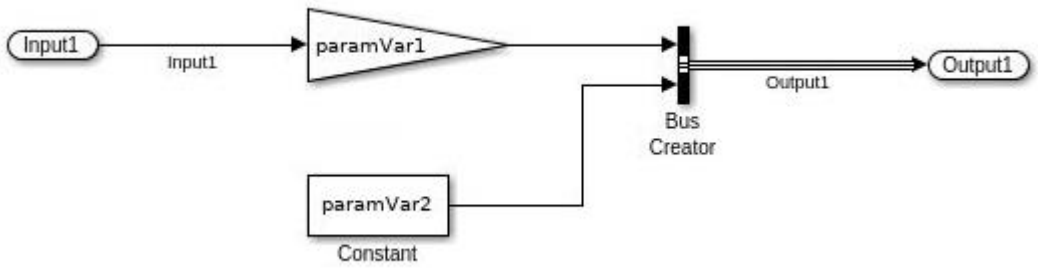
```
git clone https://vcis-gitlab.f4e.europa.eu/aneto/MARTe2.git MARTe2
git clone https://vcis-gitlab.f4e.europa.eu/aneto/MARTe2-components.git MARTe2-components
export MARTe2_DIR=MARTe2
```

```
export MARTe2_Components_DIR=MARTe2-components
export MATLAB_DIR=DIRECTORY_WHERE_MATLAB_IS_INSTALLED
cd MARTe2
make -f Makefile.linux
cd ../MARTe2-components
make -f Makefile.linux
```

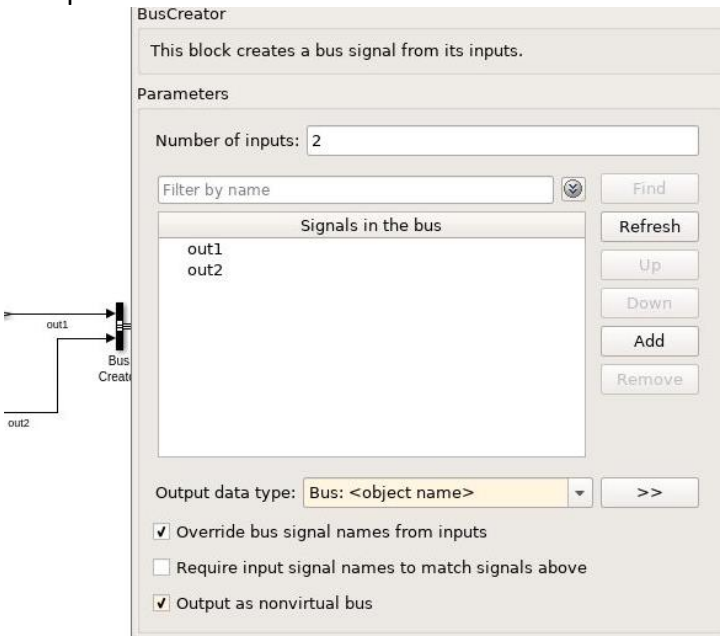
Per la configurazione della GAM vedere gli esempi approfonditi nella documentazione (file header .h della GAM)

## *Segnali strutturati (Bus)*

Sono segnali singoli con struttura annidata (come una struttura C). Anche detti bus.
Per creare un bus in uscita aggiornare il modello come segue:

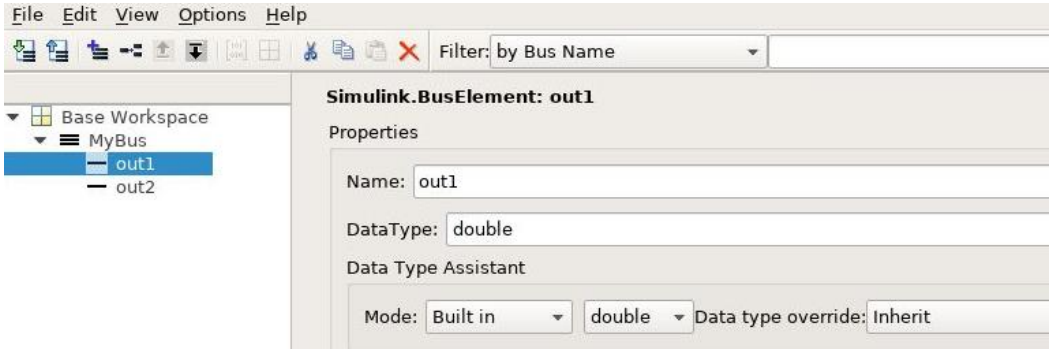



Doppio click sul bus creator, impostare output type su “Bus: <object name>” e spuntare “Output as nonvirtual bus”:



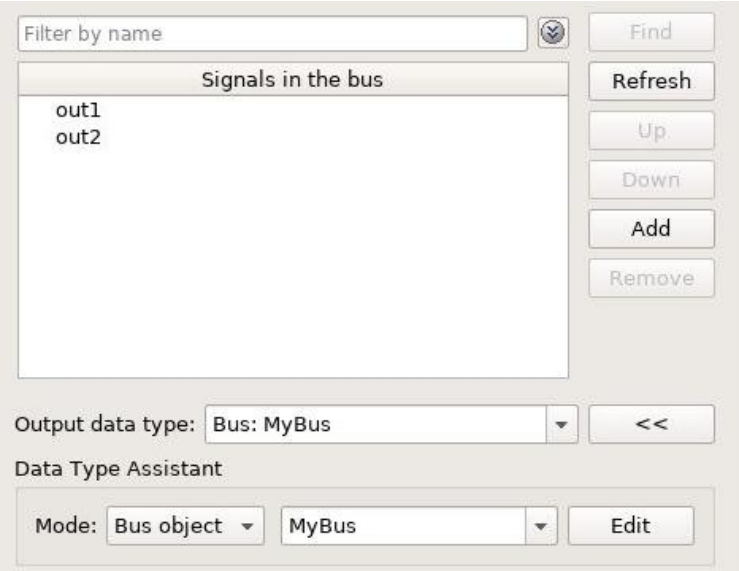

Click su “>>” e poi “Edit”:

Usare il bus editor per creare la definizione del bus (in questo caso, due segnali di tipo double scalari) che deve coincidere con il tipo e le dimensioni dei segnali che entrano nel blocco Bus Creator

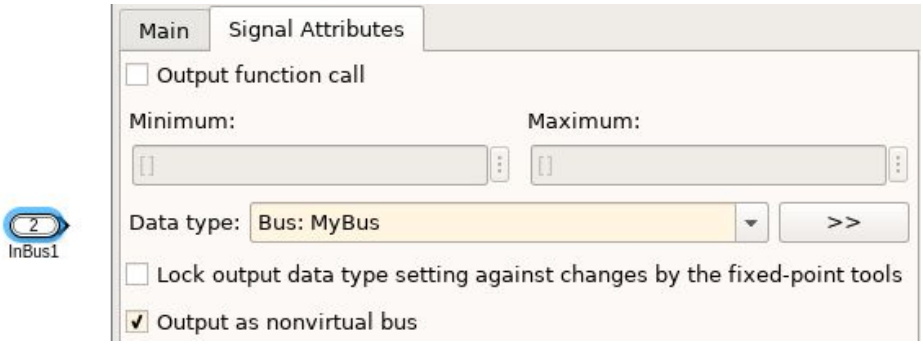



Si può esportare l'oggetto creato in un mat file (altrimenti alla riapertura del modello il bus andrà ricreato da capo).

Il bus è stato chiamato MyBus, usare questo nome nei parametri del blocco Bus Creator:




Bus in ingresso: creare un blocco "Inport", doppio click sul blocco e impostare Signal Attributes così:



Se è richiesto creare un nuovo tipo di Bus fare click su “>>” e poi “Edit”, poi procedere come prima per configurare il Bus Editor.

## *Configurazione della GAM*

Vedere la documentazione della GAM (file header .h della GAM) per esempi su come configurarla per l’utilizzo di bus.