Summer Fellowship Report

On

Creation of Block Diagrams using Xcos

Submitted by

Ranjit Roshan V S

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

June 14, 2020

Acknowledgment

I am grateful to everyone who has helped me in completing this project successfully. I would like to thank **Prof.Kannan Moudgalya** and the entire FOSSEE Team at IIT Bombay, for providing me with this wonderful opportunity to work on this project. I would like to thank my mentors **Mr.Sunil Shetye** and **Ms.Dipti Ghosalkar** for being of constant support and pointing me out in the right direction. Their guidance have been a key in helping me complete the project well within the stipulated deadline.

Contents

Chapter 1

Introduction

1.1 Objective

Xcos is an open source graphical editor tool to simulate hybrid dynamical systems which comes as a part of the Scilab package. The aim of this project include creation of block diagrams to test the Xcos-on-cloud platform developed by the FOSSEE Team and develop block diagrams for completed Text Book Companions (TBC).

1.2 Approach

For both Xcos-on-cloud and TBC's, initially block diagrams are created and run locally in either Scilab 5 or Scilab 6, which is selected based on block functionality since Scilab 6 is still not completely stable. Then these diagrams are imported and tested on the cloud platform in .xcos format. The cloud platform has option to import and run any dependency files such as functions coded in Scilab scripts.

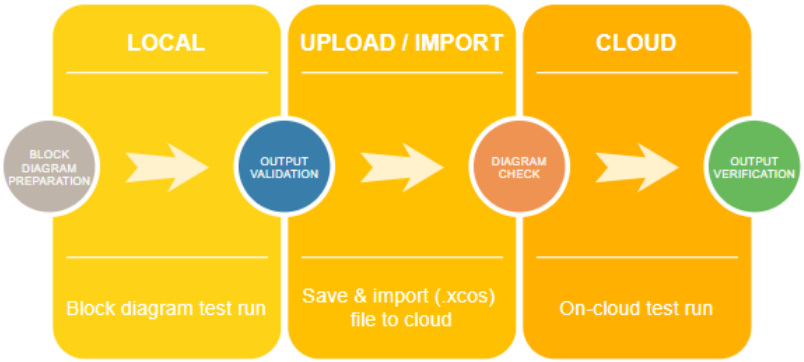


Figure 1.1: Workflow chart

Chapter 2

Technical Specifications

2.1 Xcos

Xcos is a graphical editor for simulating mechanical, thermal, electrical and control systems. It comes as a part of the Scilab numerical software tool and is being maintained by the Scilab team in France. The stable version is Scilab 5.5.2 and latest release is Scilab 6.0.2 which contains minor bugs and functionality issues.

2.2 Xcos on Cloud

Xcos-on-cloud is a cloud based simulation tool developed by the FOSSEE Team to create and simulate mechanical, thermal systems etc on a browser without the need to download and load the large tools locally. Also the cloud based simulation restricts the need for everyone to install newly released additional dependency or toolboxes each time thus saving lots of space.

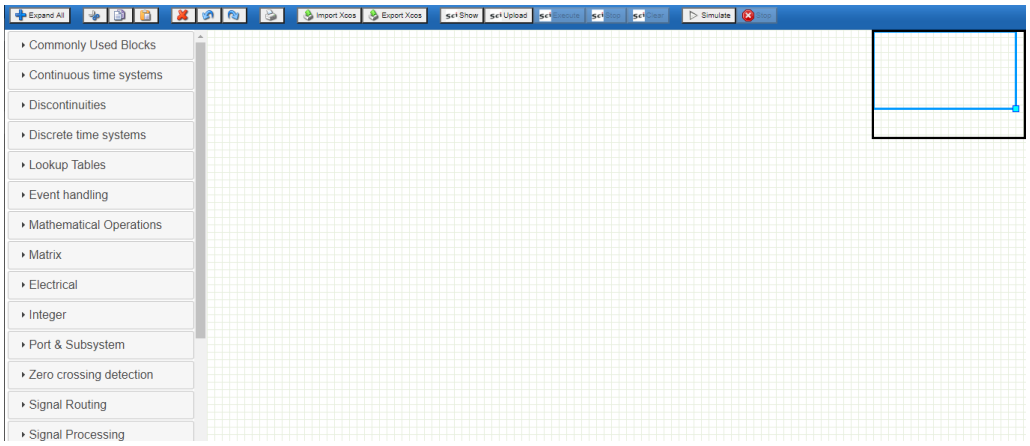


Figure 2.1: Xcos on cloud platform

Chapter 3

Xcos simulation tool

3.1 History

Xcos is based on a tool known as Scicos. Almost all the features is Xcos comes from Scicos. But unlike scicos, the latest Scilab release makes it possible to embed the Xcos and scilab onto a cloud platform through an API.

3.2 Structure in Xcos

The Xcos editor is based on two functions primarily as shown in the figure below. The *Interfacing Function* is used to configure the block related data such as

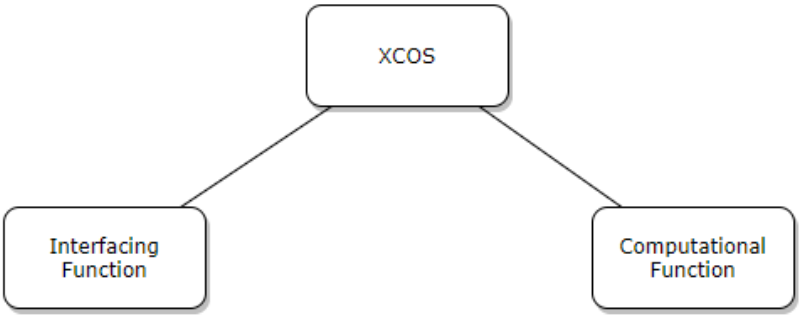
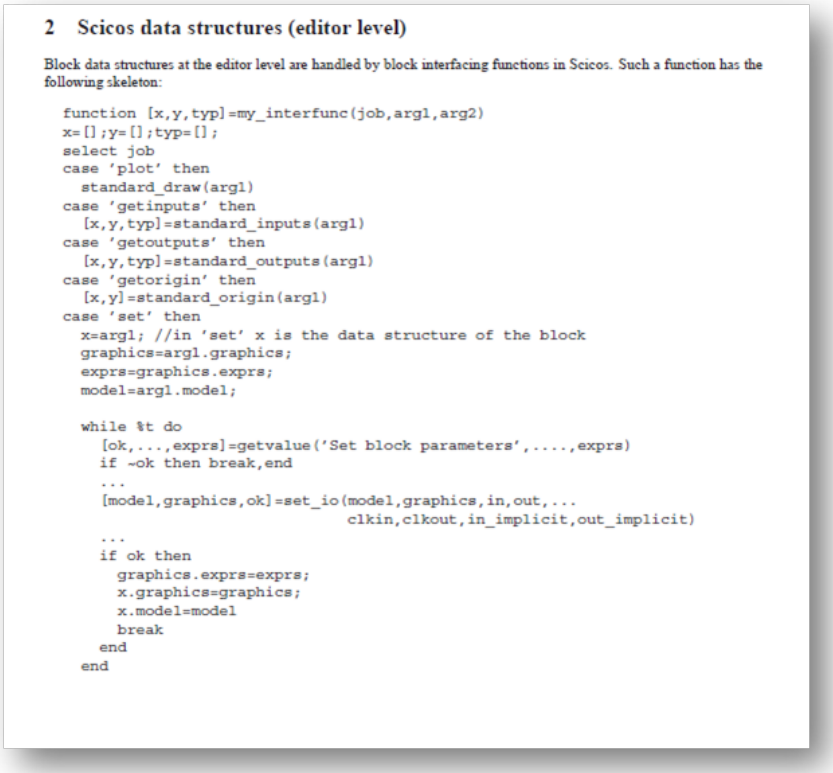


Figure 3.1: Xcos Structure

number of data inputs and outputs, event inputs and outputs, internal parameters such as gain, power of etc for mathematical blocks and so on. Based on the user inputs the port graphics are modified on screen.
The *Computation function* forms the bridge between the input data and parameters to the mathematical formulation and calculation which is programmed and compiled either in native Scilab script or in C language.

3.3 Pointers

The knowledge of pointers is needed to understand and implement custom user functionality using C Blocks. This description will also be used to explain the



2 Scicos data structures (editor level)

Block data structures at the editor level are handled by block interfacing functions in Scicos. Such a function has the following skeleton:

```
function [x,y,typ]=my_interfunc(job,arg1,arg2)
x=[];y=[];typ=[];
select job
case 'plot' then
  standard_draw(arg1)
case 'getinputs' then
  [x,y,typ]=standard_inputs(arg1)
case 'getoutputs' then
  [x,y,typ]=standard_outputs(arg1)
case 'getorigin' then
  [x,y]=standard_origin(arg1)
case 'set' then
  x=arg1; //in 'set' x is the data structure of the block
  graphics=arg1.graphics;
  exprs=graphics.exprs;
  model=arg1.model;

  while %t do
    [ok,...,exprs]=getvalue('Set block parameters',....,exprs)
    if ~ok then break,end
    ...
    [model,graphics,ok]=set_io(model,graphics,in,out,...
                               clkin,clkout,in_implicit,out_implicit)
    ...
    if ok then
      graphics.exprs=exprs;
      x.graphics=graphics;
      x.model=model
      break
    end
  end
```

Figure 3.2: Sample Interfacing Function Code

work done to create block diagrams using C Block from User defined functions in Xcos Palette in later sections.
All blocks in Xcos use the computational and interfacing functions. Pointers come into play in the computational function. This function is defined in a *structure* (similar to struct in C programming) known as **Block Structure**. The computational function inheriting this structure executes its task by pointing and calling to the variables *u1(input) and *y1(output) etc. This naming of u1 and y1 is strictly followed in all the blocks in older versions and name has changed in time to inPtr (Inputs Pointer) and outPtr (Outputs Pointer).

3.4 Structure Evolution

Earlier versions of Xcos used direct passing of all variables and parameters using what is known in C language as Call by Reference. Over time the structure method was implemented and evolved twice with *Block4* structure being the latest in use. *Macros* provide the possibility to call and manipulate block input, output data and events using functions. The figure 3.3 shows the *Block4* usage in the user defined CBLOCK4 block.

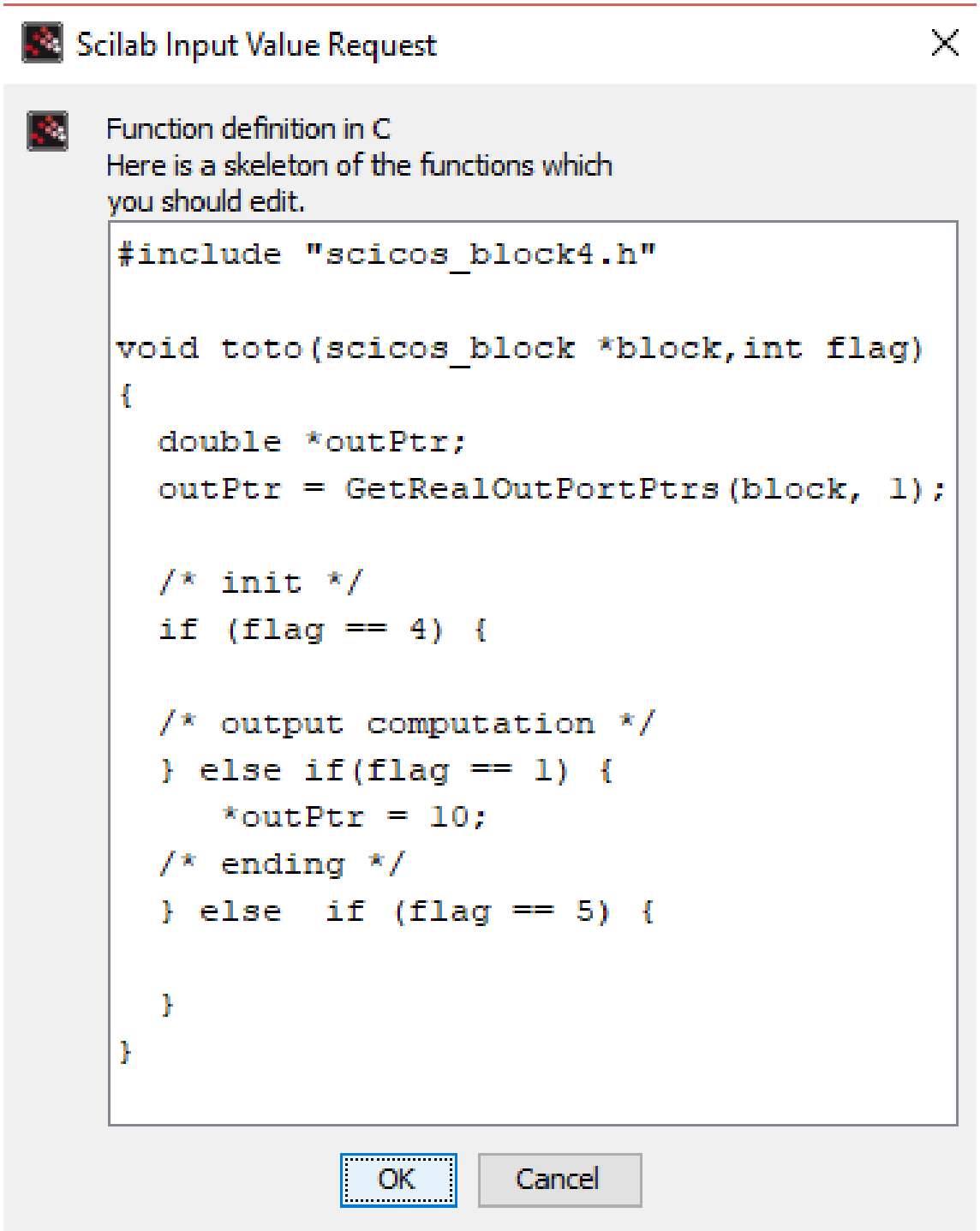


Figure 3.3: C Block struct example

Chapter 4

Xcos-on-cloud testing

4.1 Block diagram building

The block diagrams are built to test the functionality of the block completely. The parts of a block which were tested successfully are

- Computation / Math functionality
- Variable support for parameters
- Variable type
- Graphics - Rotation

The blocks are developed in a simple fashion (one/two input(s), testing block and one output)to keep the testing process efficient and this will also aid new users try and learn from these simple diagrams.

4.2 Cloud testing

Once the blocks were built they were run locally and the solution was verified. The verification process is direct observation since all diagrams either use constant inputs or sine waves and single step mathematical operations (addition, scaling etc) to arrive at the output.

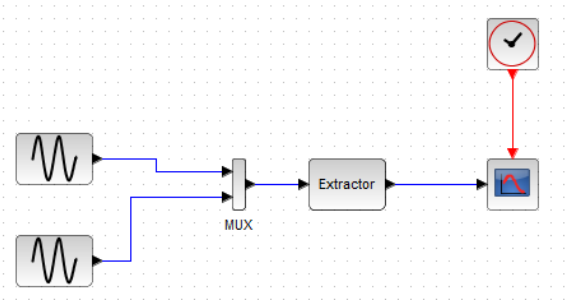


Figure 4.1: Extractor block diagram

4.3 Bugs discovered

During testing bugs relating to block functionality, block graphics, Xcos editor functionality etc were discovered. Below is a detailed description of them.

- **Invisible connected links**
  Closing the simulation and reopening is a fundamental need in any software. During reopen connection and connection ports are missing and in case of making a connection, error window pops up indicating the connected status.

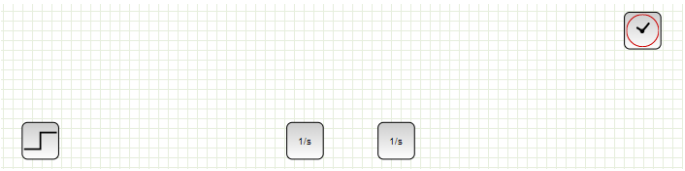


Figure 4.2: Links missing after import

- **Infinitely long connections**
  Similar to connection links disappearing, in some cases they seem to become infinitely long.

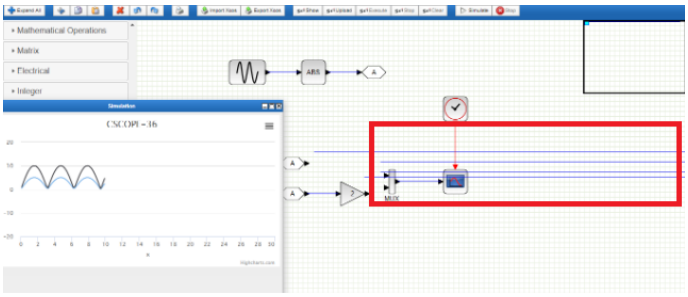


Figure 4.3: Infinitely long links

- **Rotation Problem**
  When rotated the blocks having graphic symbols other than text pops out.

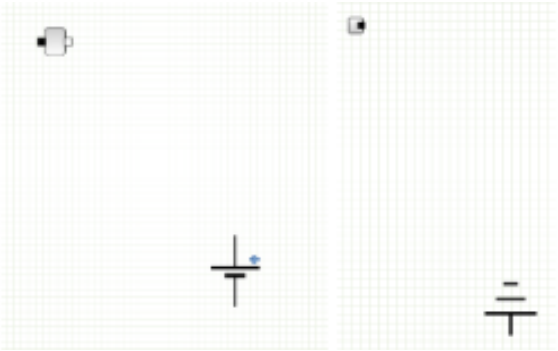


Figure 4.4: Bug in Rotation

4.4 Lack of documentation blocks

Unlike errors in functionality discussed above, there are many blocks in Xcos which lack in proper documentation and examples.

- CBlock
- CBlock2
- CBlock4
- MBlock
- General func
- Fortran Block
- PDE Block

During the course of the blocks designing, these above mentioned blocks have lack of or no documentation. Similar to Scifunc block, all the C Blocks provide very good functionality for coding in C language and integrating it in Xcos environment. (Please refer to the *Pointers* section above to know more about the C Block implementation and *References* on how to code a CBlock in Xcos)

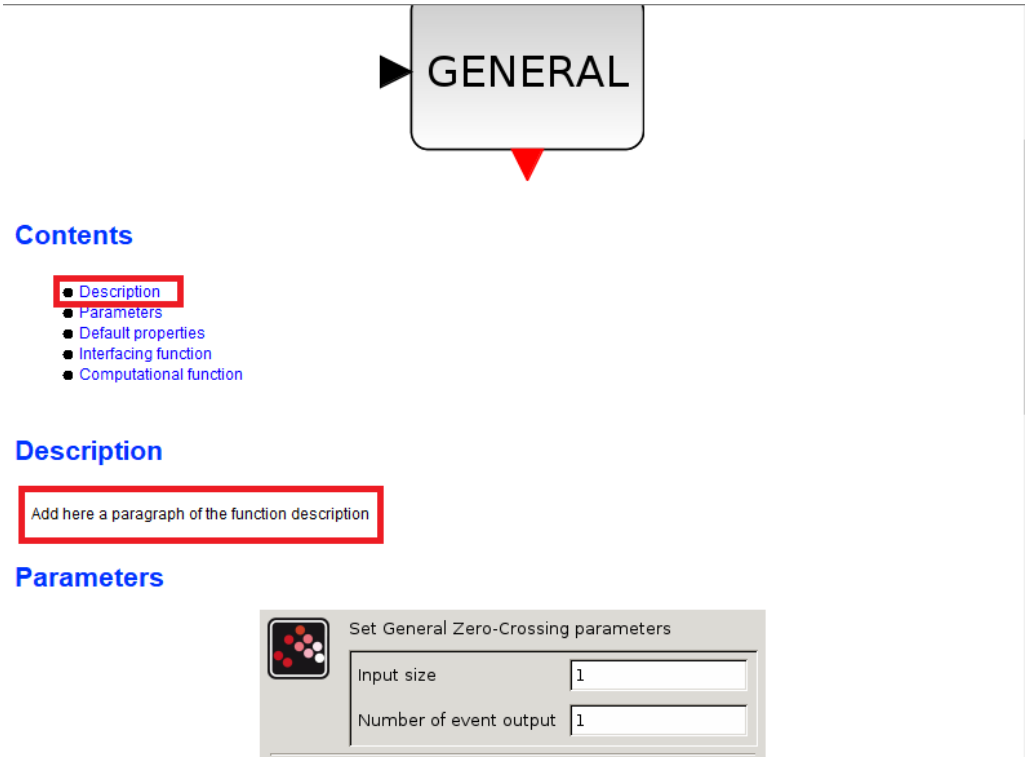


Figure 4.5: No documentation in help window

4.5 List of blocks tested on cloud

- Extractor
- Sine function
- Step Input
- CScope
- Clock c
- FROMMO
- GOTOMO
- CONST
- FROM
- GOTO
- Affich m
- General f
- TCLSS Jump
- GotoTagVisibility
- MUX
- Gain Block
- Counter
- GotoTagVisibilityMO
- Potential Block
- Vsource AC
- GND block
- Sine Voltage
- ClkInV f
- Summation
- SuperBlock
- ClkOutV f
- AND Event Block
- SampleCLK
- CEventScope
- Multipe Frequency
- Delay
- ClkGotoTagVisibility
- Logical AND
- GenSQR 1
- CSCOPXY
- Random generator
- CLKFROM
- CLKKGOTO
- M Switch
- Event Select
- Const f
- Const m
- Pulse SC
- IfThenelse f
- OutIMPL f
- Const Voltage CVS
- Vvsource f
- VariableResistor
- Tk Scale
- PerteDP
- SourceP
- PuitsP
- Flowmeter
- VanneRaglante
- Super f
- Prod f
- PNP
- Switch
- Resistor
- CurrentSensor
- c block
- ReadAU f
- WriteAU f
- ReadC f
- Trash
- CBlock2
- Constraint c
- Constraint2 c
- CBlock4
- FortranBlock
- VirtualClock
- Debug
- Sawtooth gen
- Automata
- Bache
- MBlock

Chapter 5

Text Book Companion

5.1 TBC Selection

This part of the project involves selection of 5 TBC's and create block diagrams by using the books and the examples of that book present in cloud Scilab platform. Based on my engineering background, the below books were selected for creation of block diagrams.

- Theory of Machines by R.S.Khurmi and J.K.Gupta
- Turbines, Compressors and Fans by S.M.Yahya
- A textbook on Machine Design by R.S.Khurmi and J.K.Gupta
- Manufacturing Science by A.Ghosh, A.K.Malik
- Energy Management by W.R.Murphy and G.A.Mckay

5.2 Pre-simulation

The purpose of creating block diagrams / simulation is to test, understand, obtain accurate results and relieve the user from repetitive mind calculation. Initially it may take time to adjust to the new environment but in time the end user must get familiarized quickly when a new diagram is opened. Hence certain time is spent on the analysis of end user perspective and creation of a layout which will be followed through-out all block diagrams. The following points were used to create the layout,

- Ease of inference
- Altering inputs in one operation
- Easy to locate a certain section
- Monitor different values

5.3 Layout creation

The layout is split into 4 sections in general.

- Input
- Calculation
- Output
- Monitor

The image below represents the section wise split up of the block diagrams.
Section 1 is further split into *System*, *State* and *Other* parameters.

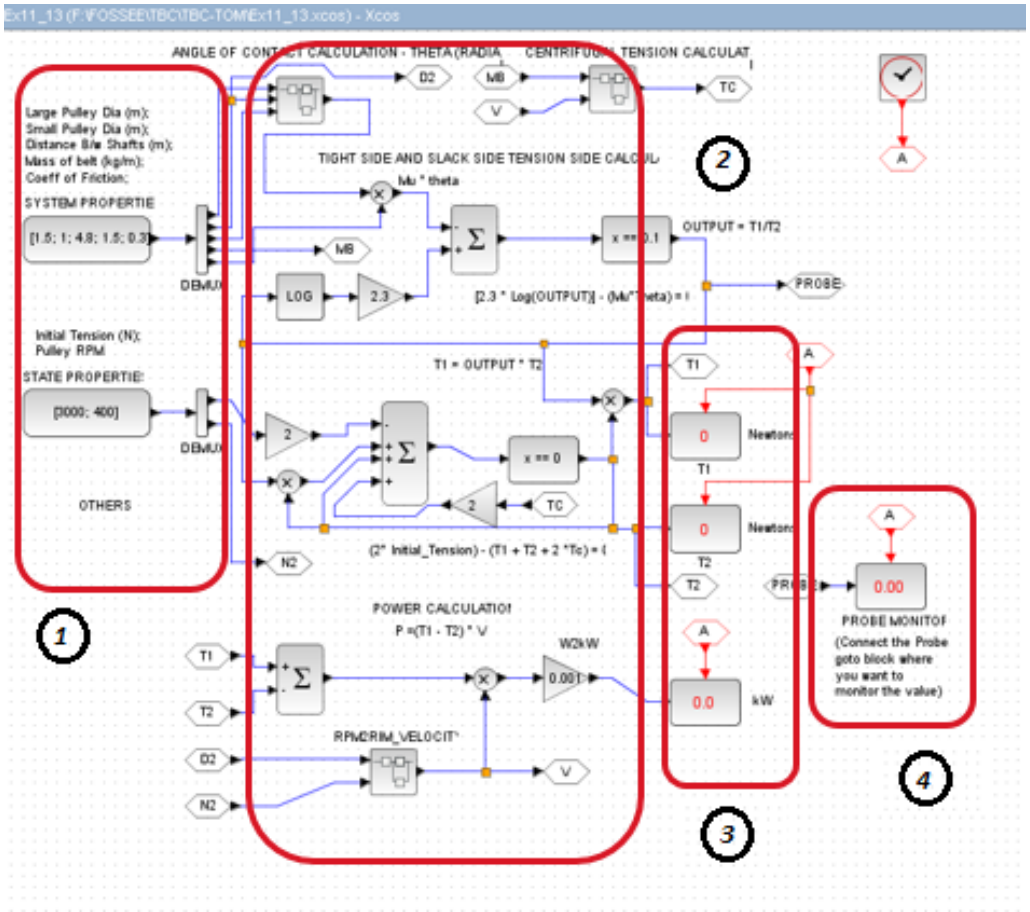


Figure 5.1: Layout of block diagrams

The System parameters refer to the internal variables such as mass, density, specific heat capacity etc.
The State parameters refer to the external variables such as velocity, acceleration, temperature etc.
Other parameters include acceleration due to gravity, gamma etc.
Section 2 includes all the formulas calculation and contains sub-headers to indicate specific sub-calculations or sub-systems.

Section 3 contains the output windows showing values or graphs corresponding to the problem requirements.
Section 4 contains blocks to monitor output values in large formulations such as fluid dynamics where the output is dependent on various parameters such as flow rate, velocity, head etc.

5.4 Simulation data

Simulation data refers to the inputs to the system taken from the TBC such as mass, density, initial velocity etc. In order to keep things in simple we keep the entire input, calculation and output on the same page.
To avoid calling from work space which uses a external script file, the constant m block is used to store all the data in the form of a row vector.

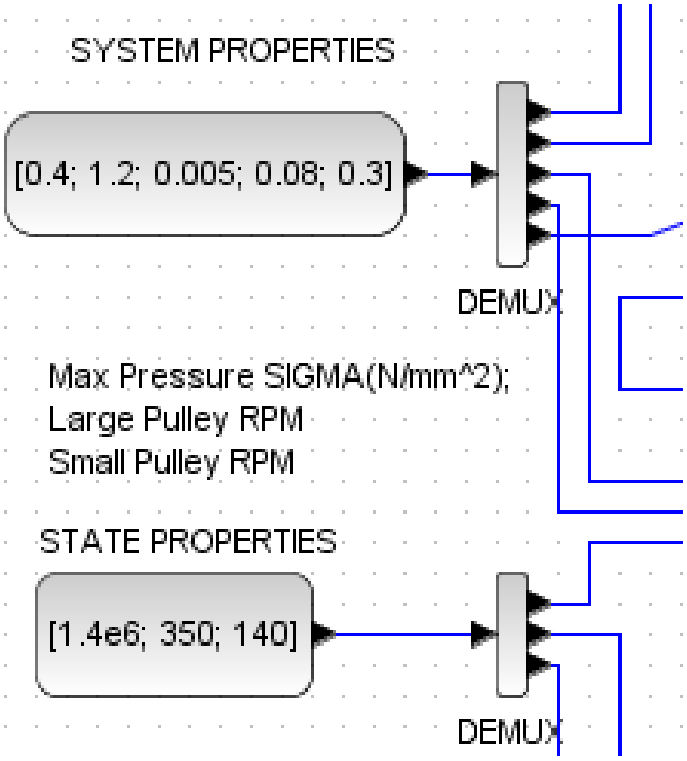


Figure 5.2: Input data

The de-mux and const m block facilitates to store and edit data in a single location as well keep the entire process of simulation in the same page.

5.5 Types of block diagrams

Over the course of creating simulation predominantly two types of diagrams are designed. Most problems come in **Explicit** form, meaning $y = x*z$ format. The other type is the **Implicit** form where the equation is of the form *y1*x*z + x/2 = 0*.

The explicit way is solved normally through getting input data, manipulating it using mathematical operators to arrive at the output.
The implicit problems is solved using the *Constraint c* block from the Implicit palette.

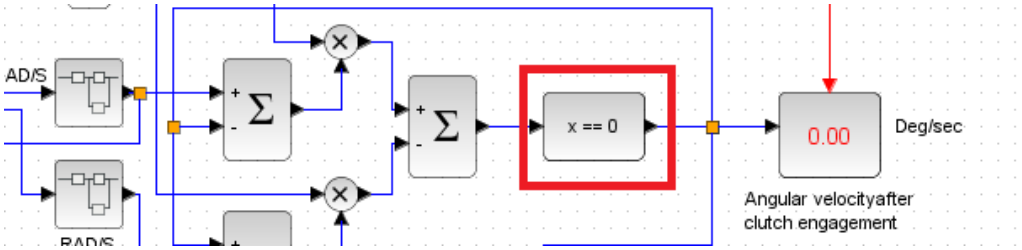


Figure 5.3: Implicit Block

5.6 On-cloud testing

The cloud testing is similar in fashion to what was done above for the Xcos-on-cloud project. The block diagram is imported in .xcos format and any dependency scripts is also loaded. All the diagrams is loaded and tested successfully.

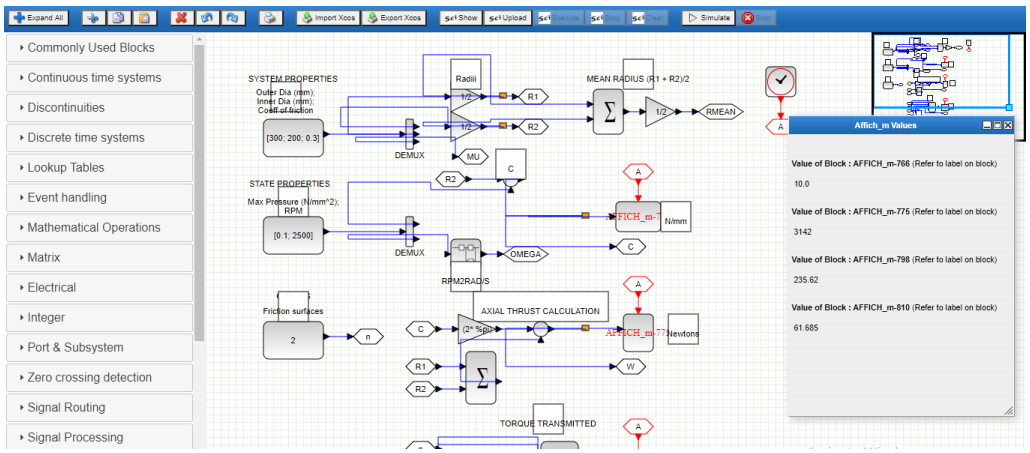


Figure 5.4: Cloud Testing

Reference

- Scilab on Cloud platform -https://cloud.scilab.in/
- Xcos on Cloud platform -https://xcos.scilab.in/
- www.scicos.org
- Learning resource -https://www.scilab.org/sites/default/files/Xcos_beginners.pdf
- Learning Resource -https://x-engineer.org/graduate-engineering/cad-cae/xcos/xcos-tutorial-simple-demo/
- https://www.rocq.inria.fr/scicos/Introduction%20to%20ScicosLab-Scicos.pdf
- CBlock4 Tutorial -https://docplayer.net/40134245-Tutorial-creating-a-c-function-block-in-scicos.html

  Threads related to Xcos block diagrams

- http://mailinglists.scilab.org/Scilab-users-Using-Modelica-generic-block-MBLOCK-in-Xcos-td4029200.html
- https://stackoverflow.com/questions/54764607/include-a-scilab-function-script-as-a-block-in-xcos-scicos
- https://users.scilab.narkive.com/A7QZC4Sc/need-help-xcos-superblock-mask-customization-error
- http://www.microdaq.org/scilab/doc/int_example.html