Model Transformation between MATLAB Simulink and Function Blocks

Chia-han (John) Yang and Valeriy Vyatkin
Department of Electrical and Computer Engineering
University of Auckland
cyan034@ec.auckland.ac.nz, v.vyatkin@auckland.ac.nz

Abstract - **This paper presents a new approach to modelling automation systems based on the combination and mutual transformation of IEC61499 Function Blocks and MATLAB Simulink. The reason for such transformation is the natural complementarity of these two models: MATLAB Simulink/Stateflow provides a nice environment for modelling and simulation of control and embedded systems, while Function Blocks are good for designing distributed control systems. An integrated software simulation environment with this transformation ability between these two tools will lead to a solution for the validation need for Function Blocks. This paper uses a motor example to demonstrate the transformation from a Simulink model to a Function Block model in Function Block Development Kit (FBDK). We discuss important issues of model transformation, such as semantics mapping, execution priorities and some guidelines for this transformation approach. An observer approach is introduced in order to make comparison of the results. This approach can also be used as part of the software simulation environment with the transformation approach.**

I. INTRODUCTION

Our earlier publication [1] has indicated that a system based on IEC61499 Function Blocks is urging for a need of a validation environment in order to deal with the complexity in the distributed design process. Due to this complexity, it is difficult to ensure the correctness and robustness of the system design. Also in the paper [2], we emphasised this importance of validating the design in a closed-loop modelling/simulation approach (**Figure 1**) and indicated that an integrated software environment for validation and verification is essential for designs based on Function Blocks. The closed-loop plant-controller modelling implies the need for systematic development of plant models in function blocks. Function Blocks are great for designing controllers based on distributed architecture with its direct deployment capability. However it is still waiting for a better software environment for design and modelling. This process is complex and resource consuming, but this problem can be solved by re-using models developed through some established modelling frameworks, or interface such frameworks from the "IEC 61499 world".

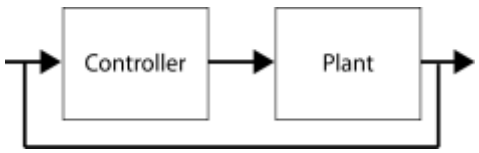


Figure 1. Closed-loop model.

Therefore, our proposed solution towards this complexity issue is by bridging Function Block model design with industrially established software tools such as MATLAB Simulink. MATLAB Simulink is a powerful tool for modelling and simulation, and is well-known by majority of researchers and industrial engineers in control system domain. MATLAB Simulink follows a block-diagram modelling approach which is very similar to the modular design approach of Function Blocks. This link between two software tools may potentially increase the industrial recognition of the IEC61499 standard. This approach can be fulfilled by direct socket communication (such as UDP or TCP) between the tools, or a complete model transformation. Since there is still no readily usable validation tools for Function Block systems, such link can allow Function Block-based design to utilise Simulink's validation functionality (simulation and data analysis).

The work has been started with an automated model transformation of a MATLAB Simulink model based on "Stateflow" blocks to a Function Block model, as published in [1]. So this work is a follow-up from the previous work with a semi-automated approach for a complete model transformation from Simulink model (not just Stateflow blocks) to a Function Block model. The transformed Function Block models can be executed under different Function Block platforms (i.e. FBDK, nxtStudio etc) after a slight modification.

In this paper, we will focus on the transformation approach that allows model transformation from MATLAB Simulink model to Function Block model. Section II describes the model transformation in details and states the semantics issues and the importance of the execution priority. The execution semantics are carefully considered without losing the nature of the models' behaviours. Section III presents the guideline for constructing the corresponding Function Blocks. Section IV describes an observation layout used for obtaining the results. Section V presents the results following by a motor example. Section VI summarises the work and discusses some further work in this research.

II. TRANSFORMATION APPROACH

The model transformation follows a block-to-block approach. On top of the Stateflow blocks, Simulink has a library storing many blocks that are often used in modelling. For different types of system, it presents different corresponding packages.

In order to perform the transformation, a corresponding Function Block library (**Figure 2**) for all the Simulink blocks used in the model must be constructed. The idea is

to create a Function Block that has exactly the same inputs, outputs and parameters as its corresponding Simulink block.

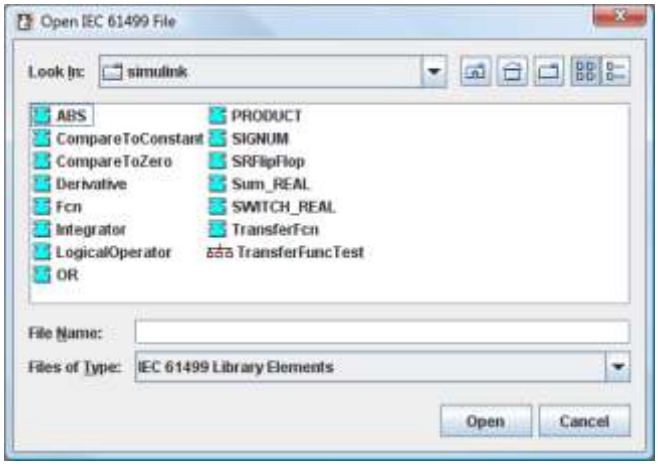


Figure 2. An example of Function Block library

A. *Semantics of the models*

Understanding the semantics of both MATLAB Simulink and Function Blocks is essential for performing model transformation between the two. Here is an overview of these two software tools.

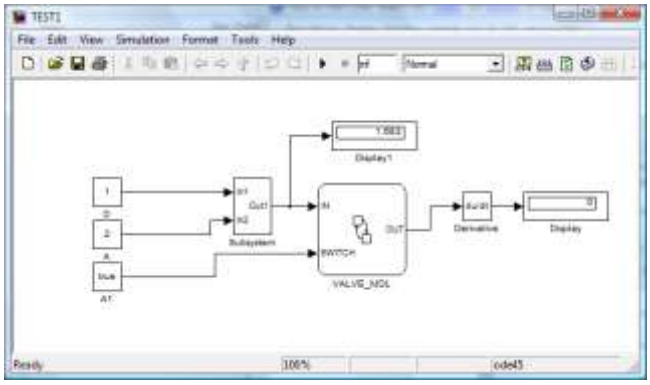


Figure 3. A simple Simulink system model

Simulink is a software package from Mathworks Inc., which provides an environment for simulation and model-based design of dynamic control or embedded systems. It is tightly integrated with MATLAB. Modelling in Simulink is done by creating a network of "blocks," stored in the Simulink library. One example of a Simulink model can be found in **Figure 3**. This example contains not just a normal Simulink block but also a Stateflow block and a subsystem block. Simulink supports simulation for discrete, continuous and even hybrid systems through its corresponding solvers. It also supports hierarchical structuring of models by grouping the related blocks into "subsystems."

Simulink solver computes the states of the system at successive time steps over a specified time span. This time steps can be fixed or variable. Simulink provides a set of solvers, where each follows a specific approach in solving a model [3]. Fixed-step simulation is where the states of the system are calculated at fixed intervals. Variable-step simulation relies on the solver to determine the length of the time steps, and the time steps will vary over time. When the system variables are changing more rapidly, the step size will be decreased, and vice versa. The solvers are generally categorised into continuous solvers and discrete solvers. Continuous solvers compute the state of a system in the current time step by using numerical integration from the state of the system in the previous time step and the state derivatives. Discrete solvers primarily solve only discrete models. They rely on the model blocks to update the discrete states of the models. There is no single method that can solve all types of models. More detail information can be found in [4].

As mentioned previously, IEC61499 standard [5] introduces Function Blocks as a new modular way of designing controllers and modelling control systems. It is believed that modelling systems with a Function Blocks network will improve the flexibility, software reusability and reconfigurability in distributed control systems design [6]. **Figure 4** shows an example of a function block model of two tank system with a pump and a valve. There are two controllers in this example where one is controlling the pump and the other is controlling the valve. The aim of this tank system is to maintain the water level of tank 1 at a specified level. The pump follows a PID control algorithm to pump water from tank 2 to tank 1 if the water level of tank 1 is lower than the specified level. The valve will direct the water from tank 1 back to tank 2 if the level is higher than the specified level.

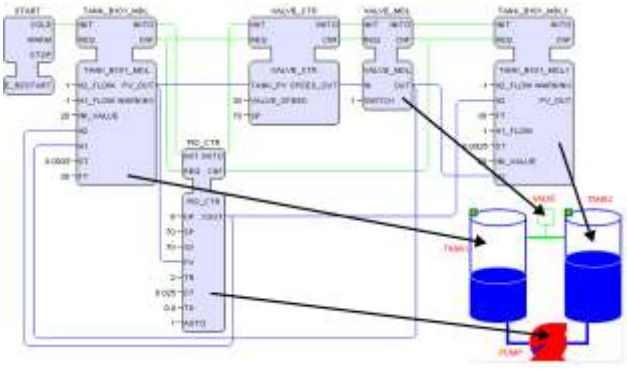


Figure 4. An example of FB network for a 2-tank system

The IEC 61499 standard defines a few design artefacts, such as basic and composite function blocks. A basic Function Block is a single event-driven module. It contains an Execution Control Chart (ECC) which is a finite-state machine. Therefore the basic structure of a basic Function Block is very similar to a single MATLAB Stateflow block. ECC describes the conditions of transitions between states and algorithms associated with each state. An example of a basic Function Block and ECC can be found in **Figure 5**. A Function Block system supports hierarchical layout as well through the use of the "Composite Function Blocks," which describe a subsystem of the whole Function Block network.

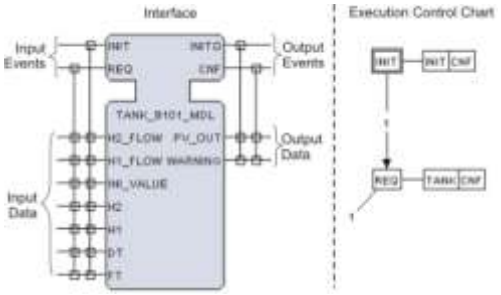


Figure 5. Interface and ECC of the basic Function Block representing a tank system.

Execution of Function Blocks is achieved by compiling them into executable code which works in conjunction with some pre-defined libraries. The code generation model plus the libraries are commonly referred to as run-time environment. One of its important tasks is to dispatch events among Function Block network. Currently, there are several different run-time environments which are implemented with different execution models. In our current exercise, The Function Block Run Time (FBRT) and Function Block Development Kit (FBDK) [7] are used, simply for demonstration purpose. The Function Block Run Time (FBRT) has the longest history in the IEC61499 community. It uses "direct function calls" where an output event triggers the successive Function Blocks in a single thread. However, this mechanism may result in stack overflow in case of event feedback loop, or in starvation of some blocks because the execution process of the caller block will be halted until all other function blocks along the event propagation path have completed execution. There is one solution to this problem by the use of LOOP_END block, which will be described later in the paper. FBRT is written in Java, and is the built-in run-time of FBDK.

B. Block-to-block mapping

The transformation must be done in a way that preserves the natures of the entire system. The behaviour of the plant must remain the same in both Simulink and Function Block models. Taking a closed-loop system for example (see **Figure 6**), both controller and plant blocks can still remain separated after transformation. The idea is to maintain the same system modelling architecture. A block-to-block transformation is the easiest way to implement this concept. With exactly the same system layout, the behaviours of the models will be the same as long as the execution order is the same.

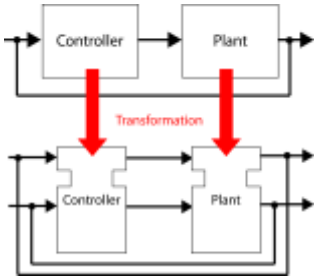


Figure 6. Closed-loop model transformation between two domains

Transforming a single Simulink block to a basic Function Block can be implemented directly as they can share exactly the same arithmetic interpretation of a model, which need to be captured in the syntax of both languages. A subsystem can be mapped to a composite Function Block. Eventually the Simulink model will be transformed into a Function Block network system. **Figure 7** is the transformed Function Block system of the example Simulink model shown in **Figure 3**.

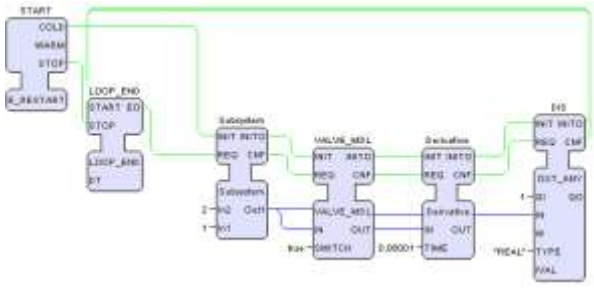
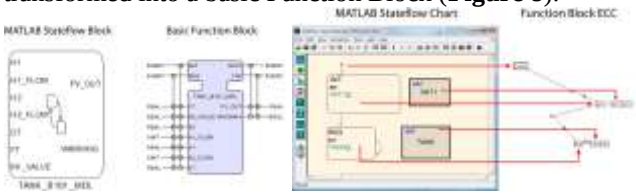


Figure 7. The transformed Function Block System

From our previous work, a Stateflow block is also transformed into a basic Function Block (**Figure 8**).



(a) **(b)**

Figure 8. (a) Stateflow block is transformed into a basic FB. (b) Stateflow's FSM is mapped with a ECC

C. Execution Priority

Simulink follows a cyclic execution order and automatically assigns execution priority of the blocks, based on some fundamental rules. Basically the rules state that a block generally has higher priority than the one where its output data is connecting to (see **Figure 9**). However there are some exceptional cases which are not really documented well in the Simulink user guide. Therefore some manual modification to the transformed model is required in these cases.

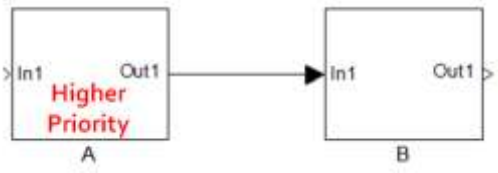


Figure 9. Execution priority in Simulink

In order to map the cyclic execution order in Simulink, a "LOOP_END" block is introduced in all the FB environment which does not follow a cyclic execution (i.e. FBDK), with all the other Function Blocks connected in a sequential order (see **Figure 7**). This LOOP_END acts just like a DELAY Function Block but with zero delay time. Because of the threading effect of the DELAY block, it manages to set a break between function calls and

therefore allows events to be connected in closed-loop in FBDK. However this DELAY block is not required in cyclic run-time environment.

III. Guidelines For Function Blocks Construction

The following guidelines are provided in order to use the transformation tool developed. Here, the Transfer Function block (**Figure 10**) is used as an example for creating the corresponding Function Block. **Figure 11** shows the corresponding Function Block created by following the guideline.

Figure 10. Transfer Function block in Simulink

Firstly, a simple database is required to allow the block-to-block mapping. The proposed database structure is shown in **Table 1**. In this database, each entry on the above table is separated by a "TAB" (or "\t"), while each line represents a single block transformation. This database information can simply be stored in a txt file, where the transformation software can just read from this file to determine the transformation decisions.

Table 1: Database structure for Simulink block transformation

Simulink Block Name	FB Name	No. of Input	Input Names	No. of Output	Output Names	No. of Internal Parameters	Parameters Names

Despite the names of the Simulink block and its corresponding Function Blocks, the input and output names must be listed, in the order according to the order inside Simulink. This is because Simulink only stores I/O port numbers but not the I/O port names in the model file. Parameter names must be also provided, because in our approach the internal parameters of the Simulink block are set as "data inputs" (instead of internal variables) of the corresponding Function Blocks (see **Figure 11**) in order to be graphically modifiable in the user interface.

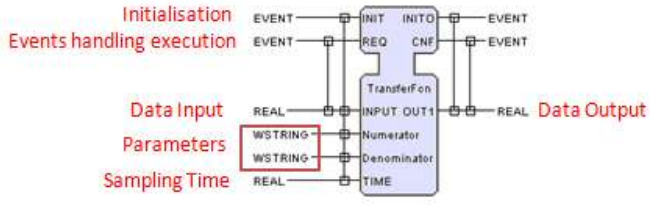
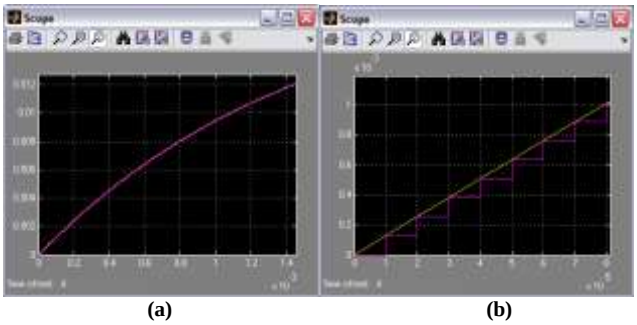


Figure 11. The corresponding Transfer Function FB

Some Simulink blocks are associated with the sampling time of the system execution especially in the continuous time modelling. For these types of blocks, a "TIME" input parameter must be given for their corresponding Function Blocks. However this sampling time parameter here indicates a discretisation of the continuous data when running the model under the Function Block environment. **Figure 12** presents the comparison of the output results of the Transfer Function Simulink block with its corresponding Function Block. The values are exactly the same only on its successive time step, followed by the sampling rate.



(a) **(b)**

Figure 12. (a) Comparing the outputs of the Transfer Function Simulink block with its corresponding FB. (b) A close-up of the results comparison.

When creating the corresponding FBs, WSTRING is used as the data type for all the "parameters" (but not for inputs and outputs). Also the naming of the blocks in the Simulink block must not contain "space" or "symbols."

All the Function Blocks will be created with an initialisation event I/O, associated with all the parameters and data I/O, and also a REQ/CNF event I/O pair for executing them in the correct priority order, associated with purely data I/O.

IV. Observer Layout

In this exercise, an observer layout has been introduced in order to compare the results of the transformed Function Block model with the original Simulink model, by the use of socket communication protocol (i.e. UDP or TCP) with full handshaking at every sampling time. There are two ways of using this observer layout. One is to directly compare the output results of the transformed Function Block model with the original Simulink model. Simulink has readily well designed interface for graphic display, so the output data of the Function Block model is sent to the Simulink environment for comparison. Then by the use of "Scope" block in Simulink, all the output data from both models can be compared graphically or in data form. An example layout is shown in **Figure 13**.

Second one is used when only a subset of the Simulink model has been transformed into the Function Block model. This is very useful when only part of the original Simulink model is required for the new design under Function Block environment. This layout then allows us to test the transformed part of the model in a closed-loop manner with the remaining part of the original model. The layout can be seen in **Figure 14**.

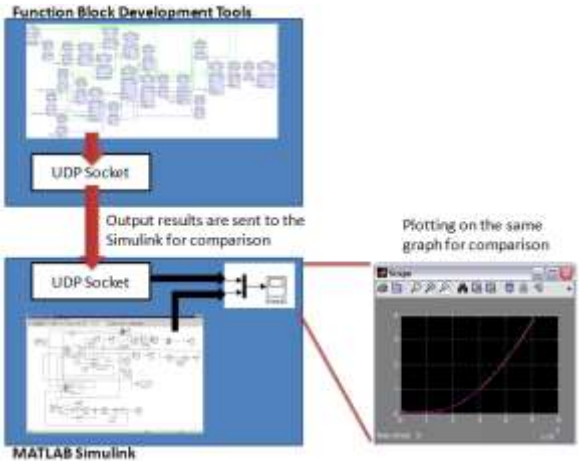


Figure 13. Compare data and simulation results by using socket communication

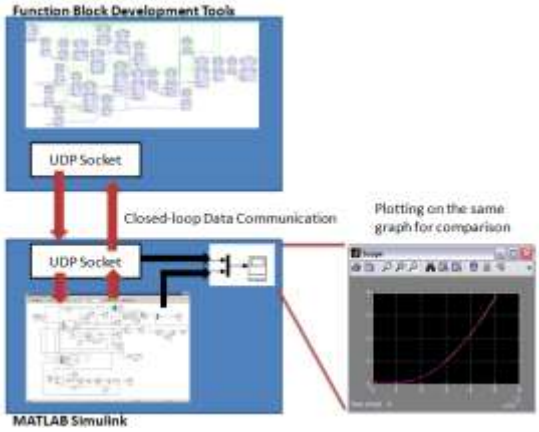


Figure 14. Closed-loop simulation with the transformed system in Function Block environment

V. RESULTS

FBDK is chosen as our Function Block development tool, even though its execution semantics are not favoured by the industry. However it is sufficient enough for academic demonstration, at such early stage of the development.

From the technical perspective, the transformation can be done with assistance of some existing APIs associated with MATLAB Simulink and Function Blocks. The work [8] provides a JAVA parser for the Simulink MDL file, and our research group has developed a Function Block to JAVA parser. Combining these two APIs together makes the development of the transformation easier.

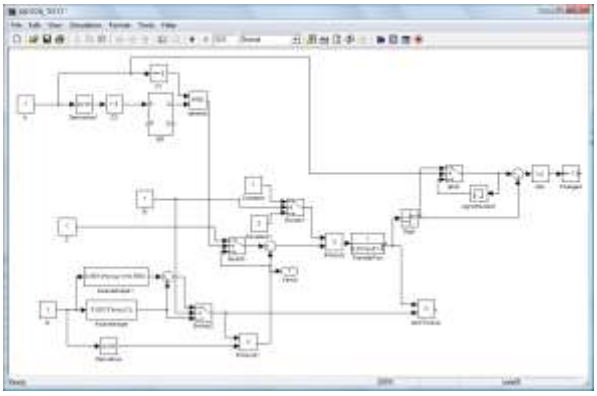


Figure 15. Motor Simulink model

A motor Simulink model (see **Figure 15**) has been experimented with this transformation approach. The Simulink model of a motor has been successfully transformed into a Function Block model that can be executed under FBDK. The resultant Function Block model after the transformation can be shown in **Figure 16**.

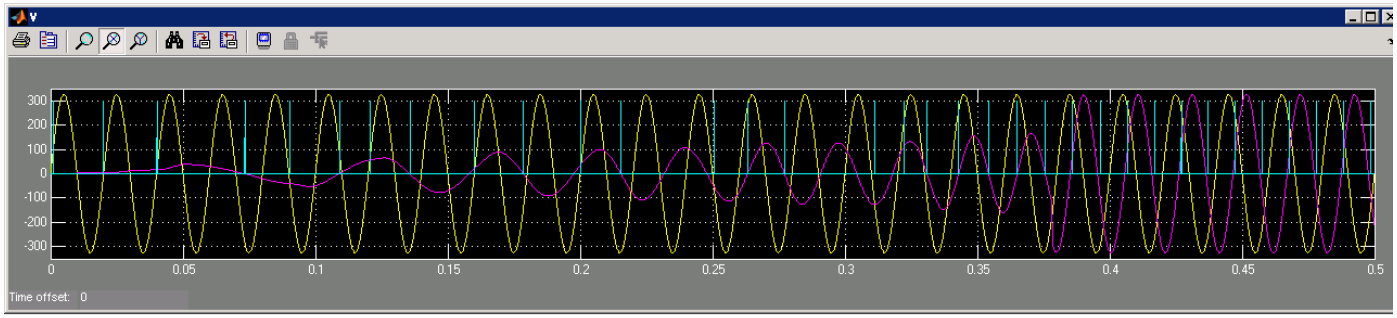


Figure 17. The original result from the motor Simulink model

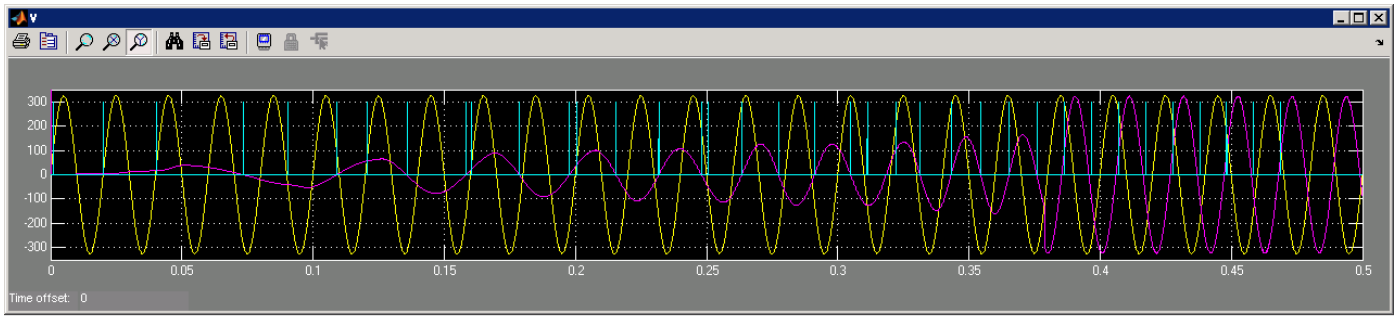


Figure 18. The output results from the transformed FB motor model

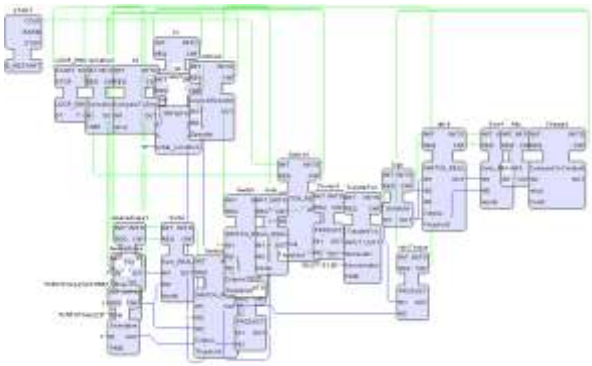


Figure 16. Trasnformed FB System of the motor model

As described before, the priority order is determined by some basic rules in Simulink. However there are some exceptional cases. Therefore the transformed model may require some modification to the priority order before producing the desired results. **Figure 19** shows the rearranged version of the transformed model.

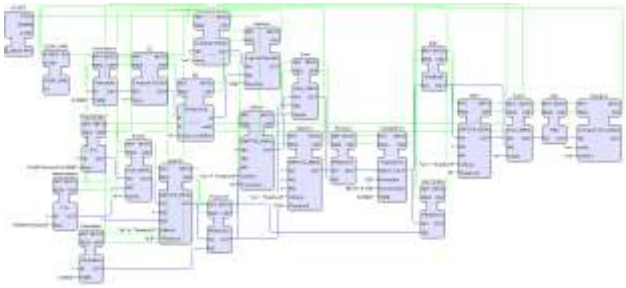


Figure 19. A "tidy-up" version of the transformed FB System

The output result from the motor Simulink model can be seen in **Figure 17**. This result from the transformed model is obtained by using the observer layout approach mentioned in the previous section. The test environment is set up as shown in **Figure 20**.

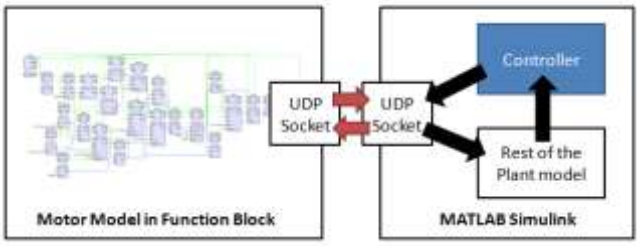


Figure 20. The observer setup for obtaining the results from the transformed FB model

Figure 18 illustrates the result~~After~~ obtained by replacing the motor part of the original Simulink model with UDP sockets to the transformed FB model.~~, **Figure 18** illustrates the obtained result~~. From inspection, the results are fairly identical where the purple line indicates the desired output.

In order to explain the slight difference in the results, a simple experiment with the derivative Function Block in the motor model has been tested with the observer approach. **Figure 21** shows the difference in the outputs of the derivative blocks. FBDK's output is in purple and Simulink's one is in yellow. Simulink's output is quite smooth and the FBDK's one has some "noise." This result can be considered as normal due to the discretization. ~~This result may~~It can be improved ~~with~~ using a more advanced modelling techniques or some averaging operation, when constructing the corresponding Function Blocks. This exercise also demonstrated the importance of the execution priority. The result varies (i.e. out of phase) if the block are executed in a completely different order. Also the accuracy of constructing the corresponding FBs is very important in this transformation approach.

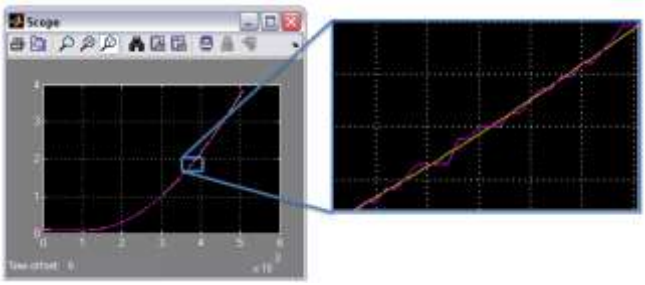


Figure 21. Result comparison of the derivative block in Simulink and FBDK

VI. Conclusions And Future Works

A transformation approach has been developed to create a simulation environment in order to help in design distributed systems with design complexity. This approach allows us to take advantage of the simulation and analysis capability of MATLAB Simulink to help in designs based on IEC61499 Function Blocks.

Further work with this transformation approach will be experimented with other Function Block tools, run-times and compilers, in order to expand the usage of the new approach with the proposed simulation environment.

References

[1] C.-H. Yang and V. Vyatkin, "Automated Model Transformation between MATLAB Simulink/Stateflow and Function Blocks," in *INCOM*, 2009.

[2] V. Vyatkin, H. M. Hanisch, P. Cheng, and Y. Chia-Han, "Closed-Loop Modeling in Future Automation System Engineering and Validation," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 39, pp. 17-28, 2009.

[3] R. Ray, "Automated Translation of MATLAB Simulink/Stateflow Models to an Intermediate Format in HyVisual," in *Computer Science Department*. vol. MSc Chennai: Chennai Mathematical Insitute, 2007.

[4] "The MathWorks - MATLAB and Simulink for Technical Computing," http://www.mathworks.com.

[5] "Function blocks: Internatinal Standard IEC61499," International Electrotechnical Commission.

[6] V. Vyatkin, *IEC 61499 function blocks for embedded and distributed control systems design*. Research Triangle Park, NC: ISA-Instrumentation, Systems, and Automation Society, 2007.

[7] "Function Block Development Kit," Holobloc inc., http://www.holobloc.com.

[8] "TUM Simulink Library," Lehrstuhl Software and Systems Engineering, http://www4.cs.tum.edu/~ccsm/simulink/.