Automatic Conversion of Simulink Models to SysteMoC Actor Networks

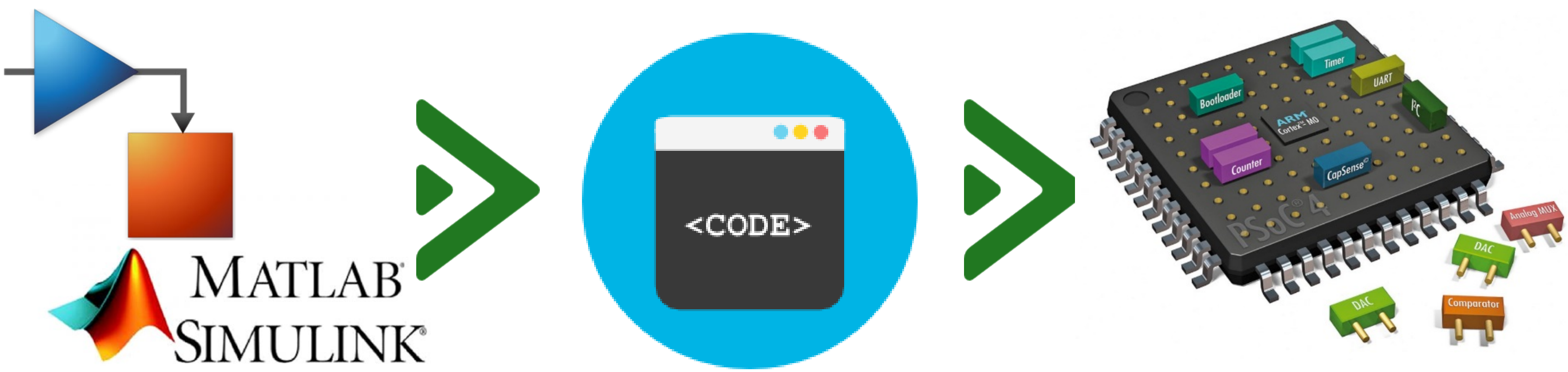
Martin Letras, Joachim Falk, Stefan Wildermann, and Jürgen Teich
Hardware/Software Co-Design, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
SCOPES'17, St. Goar, Germany, June 12, 2017

Outline

- Introduction
- Data Flow Graphs and Simulink
- Automatic Conversion Method
- Case Study
- Conclusions

Motivation

- There exist automatic translation toolchains for generation of C or C++ code from Simulink models



Source: Mathworks

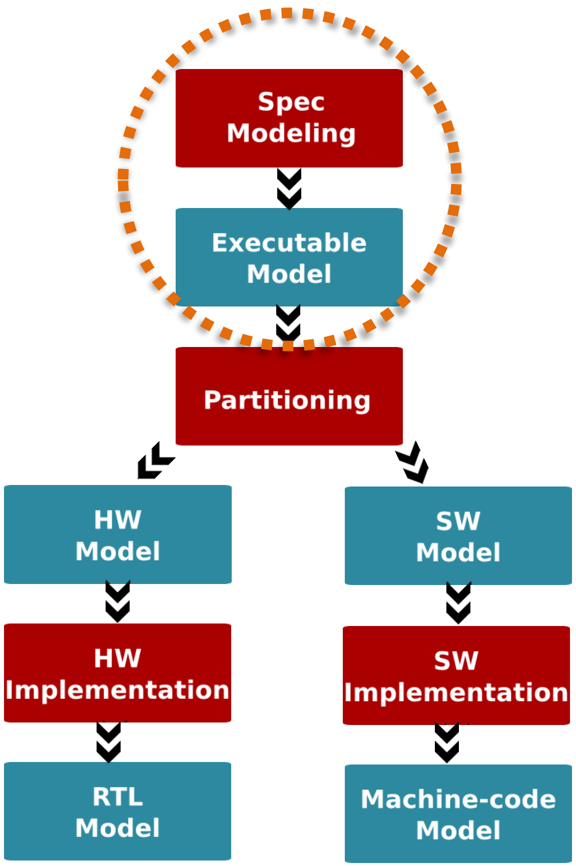
Source: Cypress

- Intuitive block-based design
- Simulation
- Rapid prototyping

- What about heterogeneous SoC architectures?

Introduction

Introduction (1)



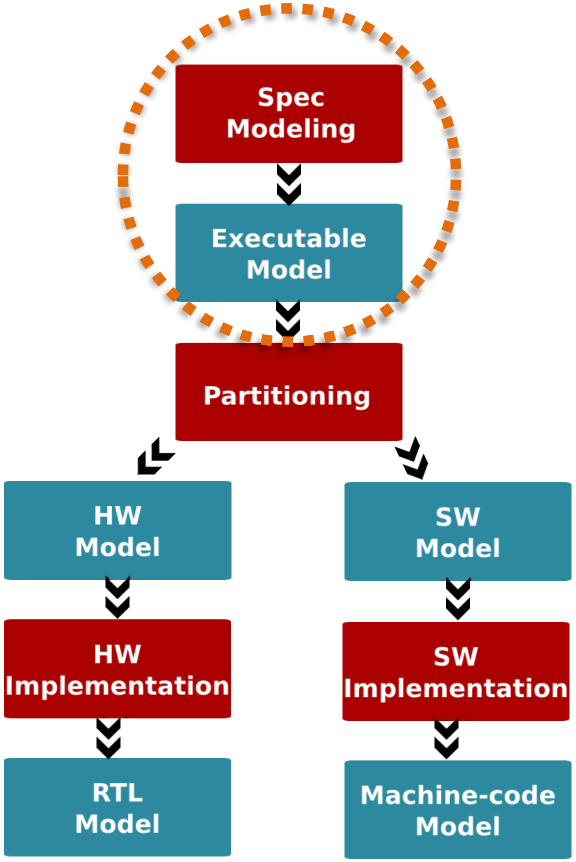
Electronic System Level (ESL) Design Methodology

- Integrate Simulink in an ESL methodology

Simulink:

- Rapid prototyping and design tool
- Mainly focused on the signal processing domain
- Toolboxes for different applications

Introduction (1)

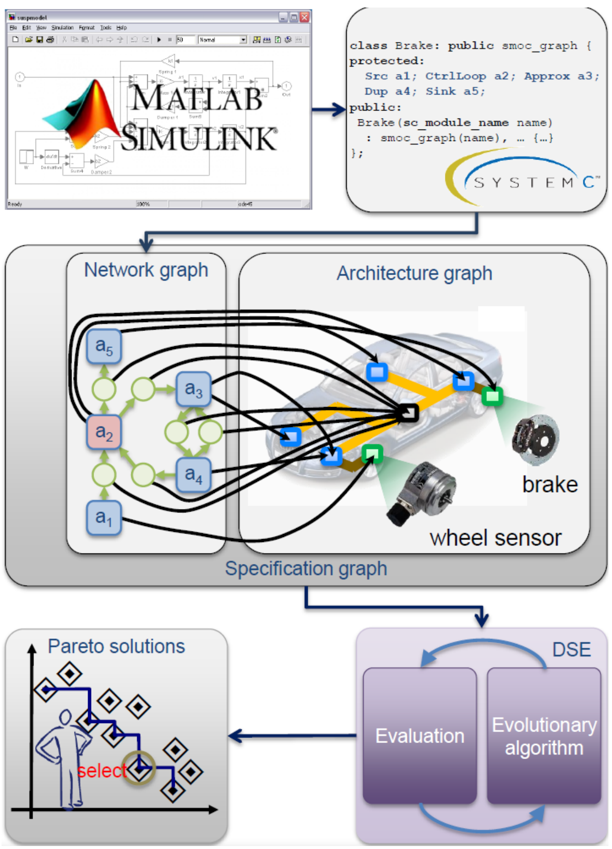


Electronic System Level (ESL) Design Methodology

- Integrate Simulink in an ESL methodology

However, automatic code generation for heterogeneous target architectures consisting of GPPs and hardware accelerators **is currently not supported by Simulink**

Introduction (3)



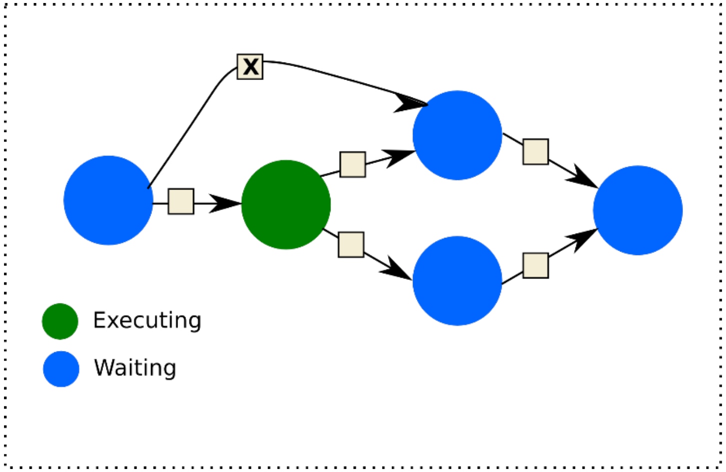
Simulink integration for SystemCodesigner

- Convert Simulink specifications to the input language of a well established ESL flow
- Facilitate hardware/software co-optimization and Design Space Exploration (DSE)
- Automatic generation of hardware/software co-designs

Data Flow Graphs and Simulink

Data Flow Graphs in SysteMoC (1)

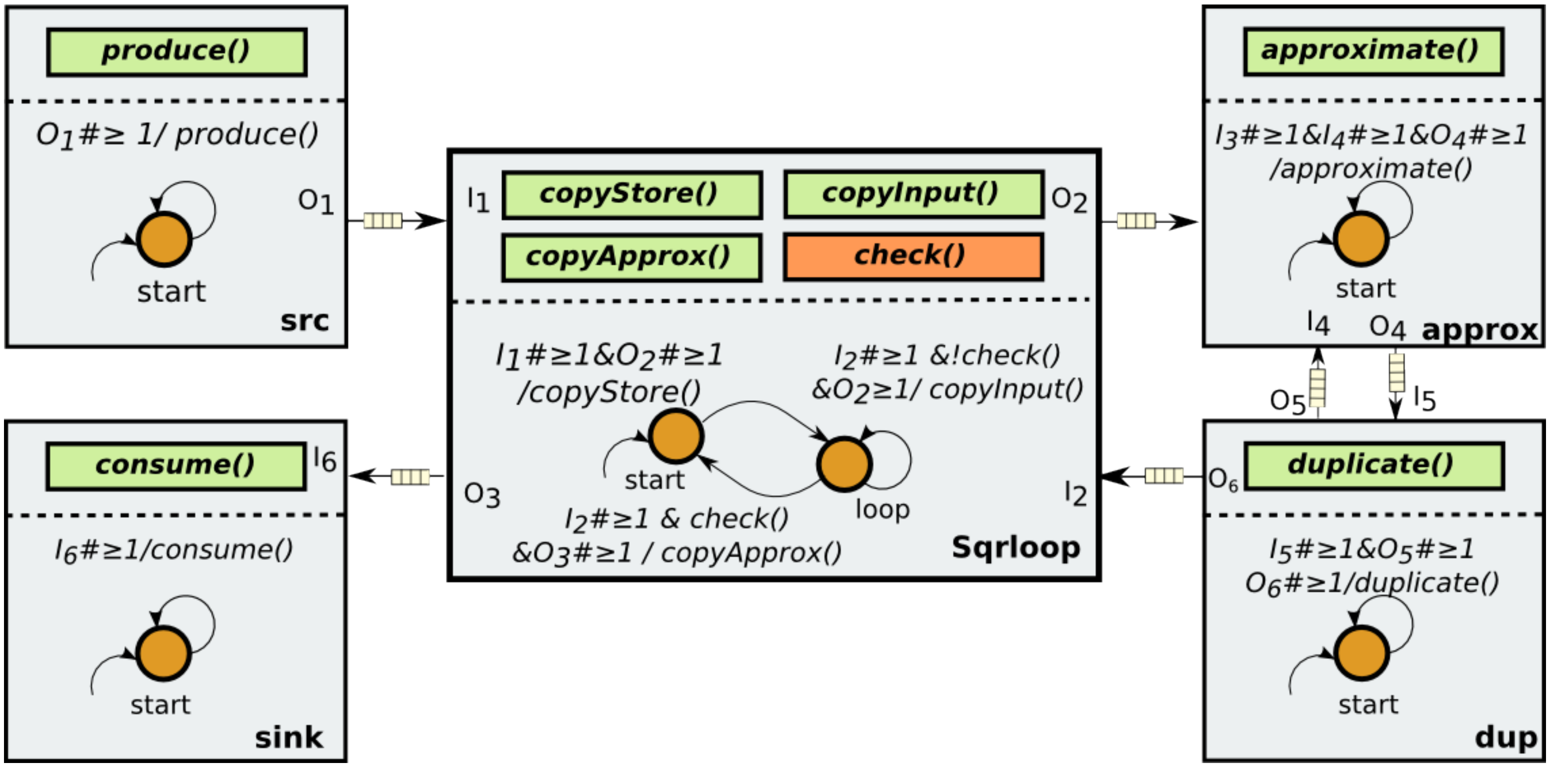
- Data flow allows modeling concurrent systems by concurrently executing actors



- SysteMoC is the input language of SystemCoDesigner

- A Data Flow Graph (DFG) is realized by a so-called actor network

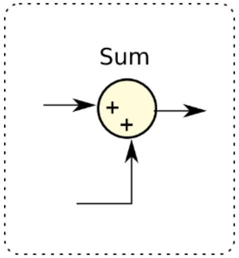
Data Flow Graphs in SysteMoC (2)



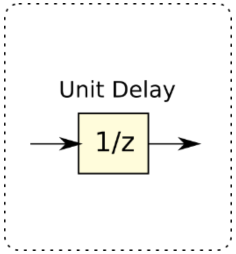
- Actors communication via channels only

Simulink

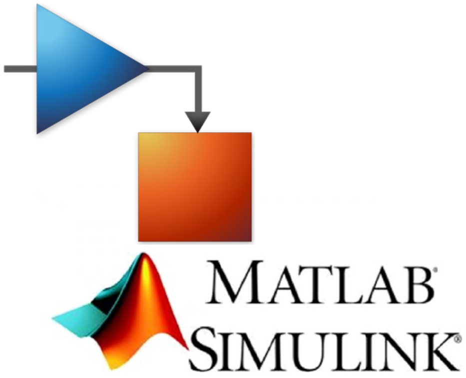
- Simulink
  - The basic elements are functional blocks



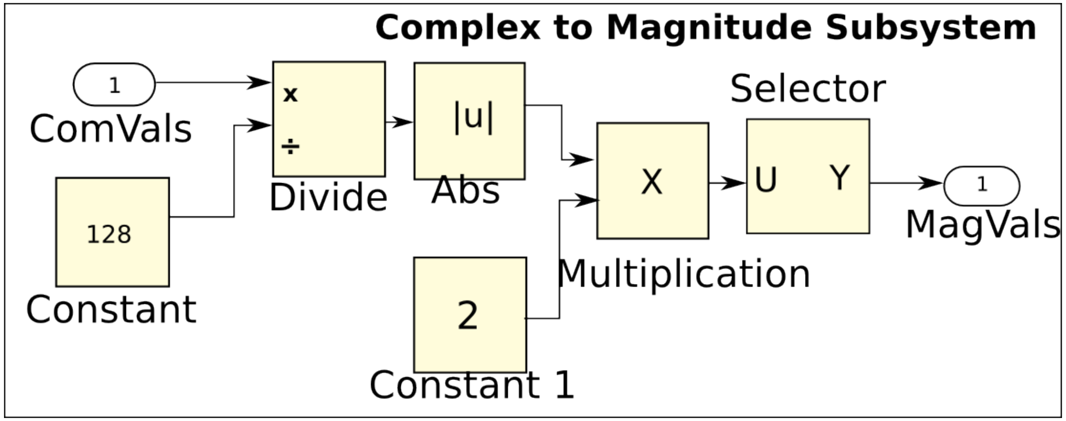
State-less block



State-full block



- The blocks can be composed in subsystems



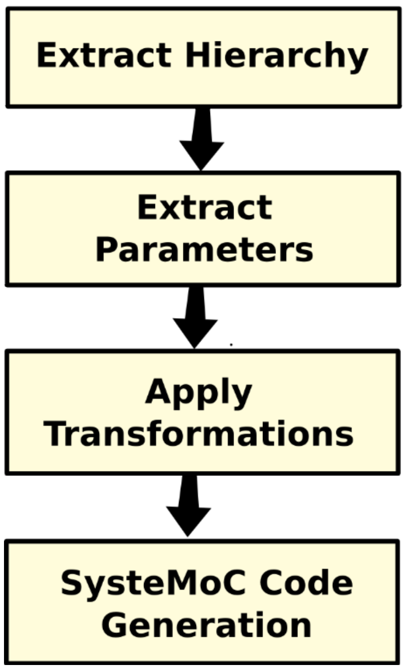
Simulink

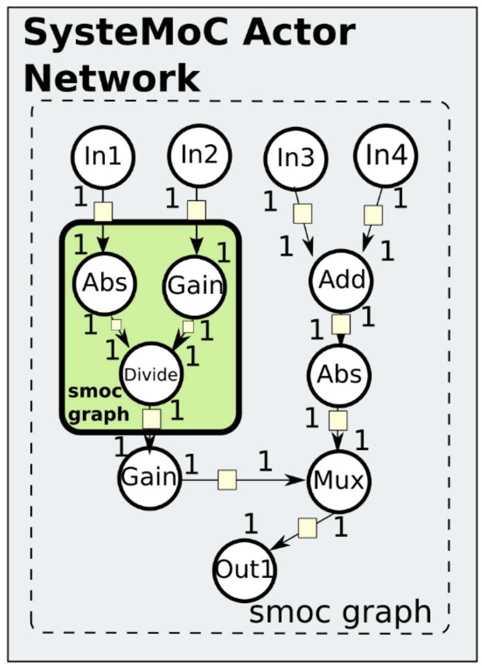
Although there are many similarities between the Data Flow Graphs and Simulink, we must have in mind the next issues:

- **Data-triggered execution vs. Time-step execution**
- **Some Simulink structures are not allowed in Data Flow**
- **Multi-rate systems**

Automatic Conversion Method

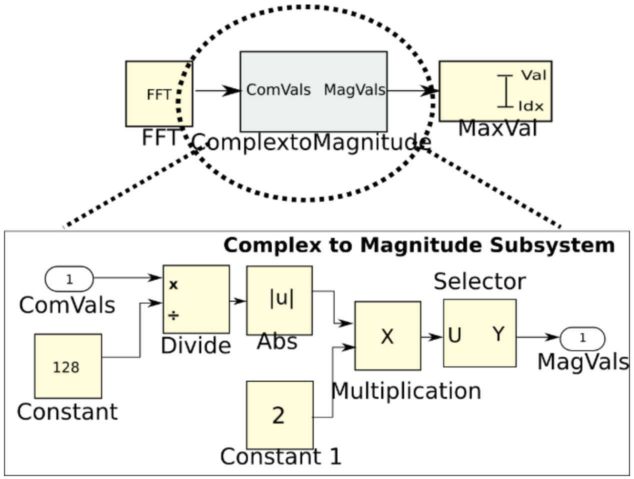
Automatic Conversion Method

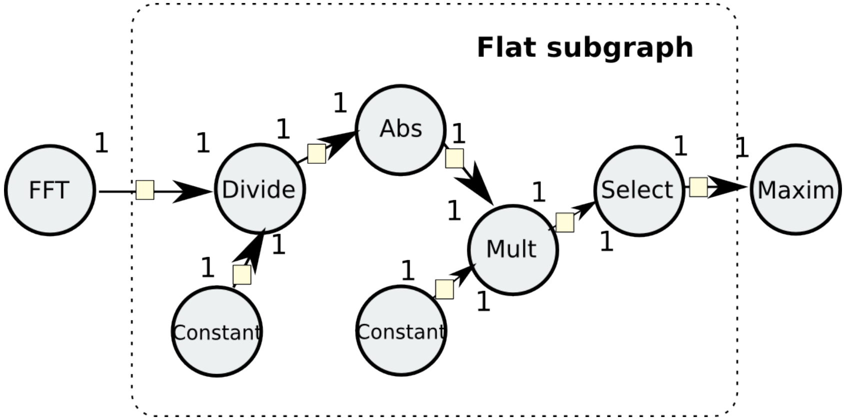





```
class:Add  public smoc_actor
{
 public:
  /* Port Definition: */
  smoc_port_in<real_T  > in_1;
  smoc_port_in< real_T > in_2;
  smoc_port_out< real_T > out_1;
  add(sc_module_name name):
         smoc_actor(name, start)
  {
    start = in_1(1)>> in_2(1)>>
              out_0(1) >>
          CALL(add::method_add) >> start;
  }
 protected:
  void method_add()
  {
     add_initialize();
     add_U.In1 = in_1[0];
     add_U.In2 = in_2[0];
     add_step();
    out_0[0] = add_Y.Out1;
  }
  smoc_firing_state start;  };
```

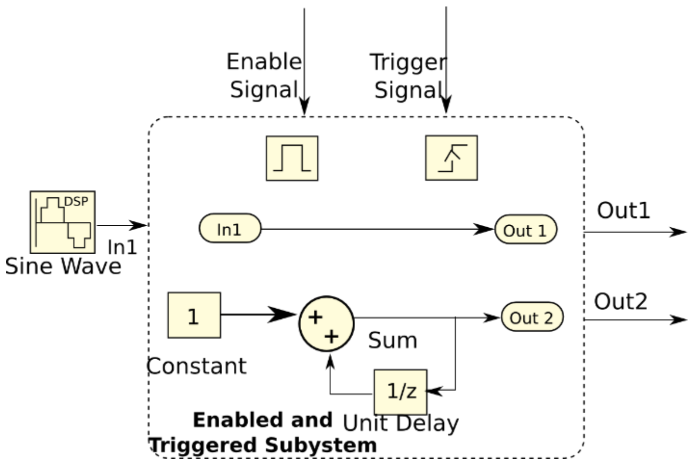
Extraction of the Hierarchy (1)

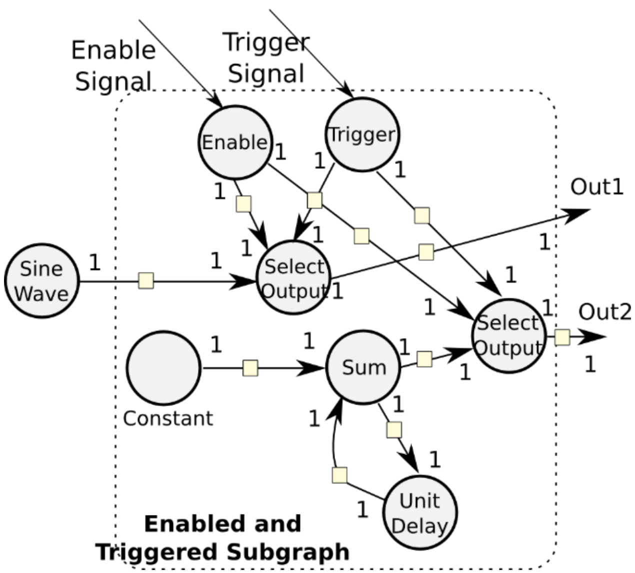




- Hierarchical structures are not defined in DFG
- Each atomic block is mapped to an Actor
- Create a flat graph for each subsystem in the Simulink

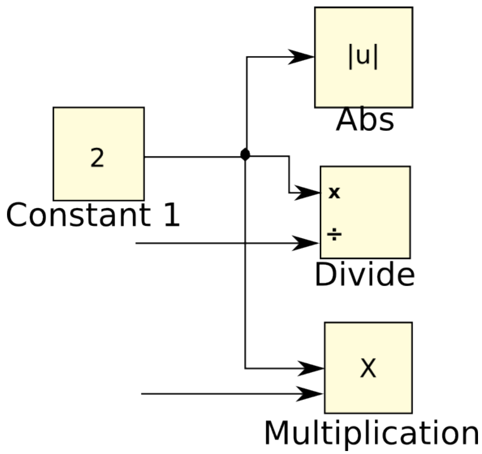
Extraction of the Hierarchy (2)

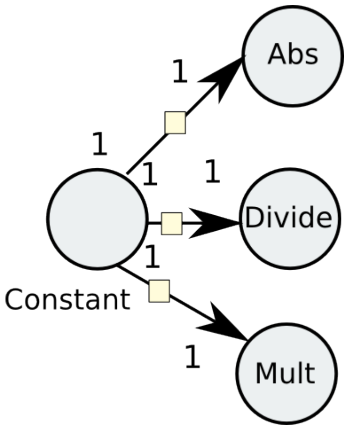




- Create a flat graph but there are two additional signals:
  - The *enable signal* determines when the subsystem is active
  - The *trigger signal* activates the subsystem according to a trigger event

Extraction of the Hierarchy (3)

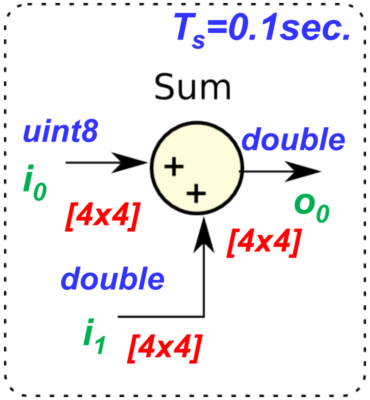




- Channels are exclusive for a source and sink actor
- Data replication on lines are converted by replicating output data

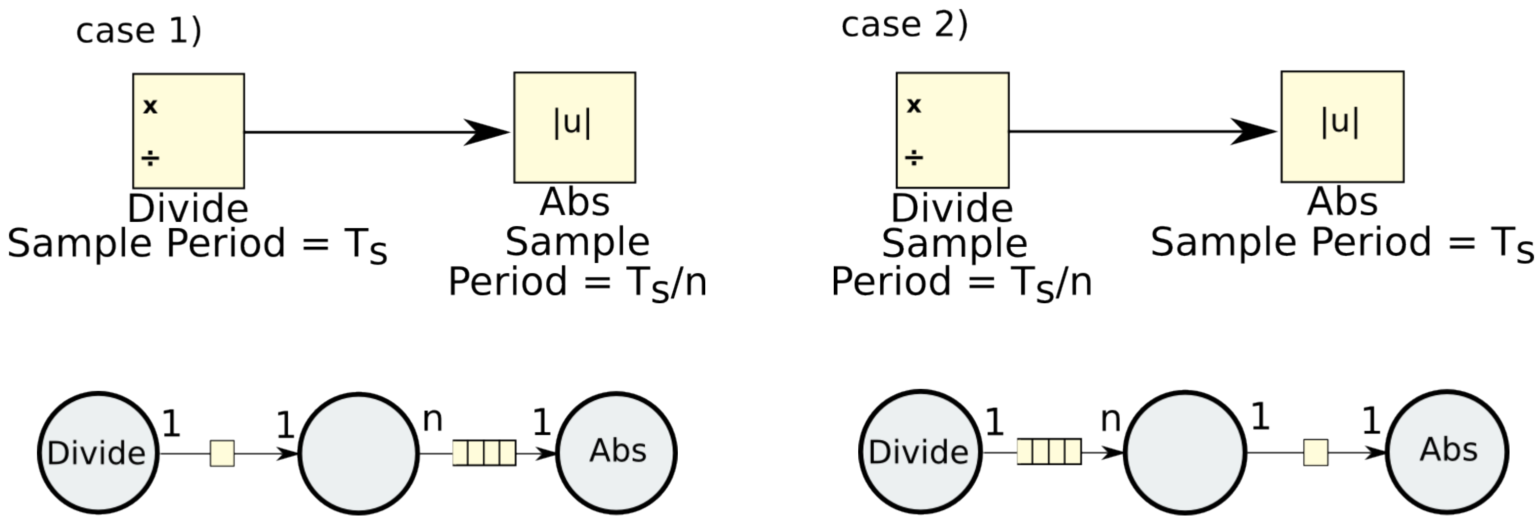
Extraction of Parameters

- Extract number of input/output ports
- Extract data types
- Resolve data types
  - Support for arrays and matrices
  - Encapsulate data in a token object
- Extraction of Sample time
- Extraction of specific parameters of each block



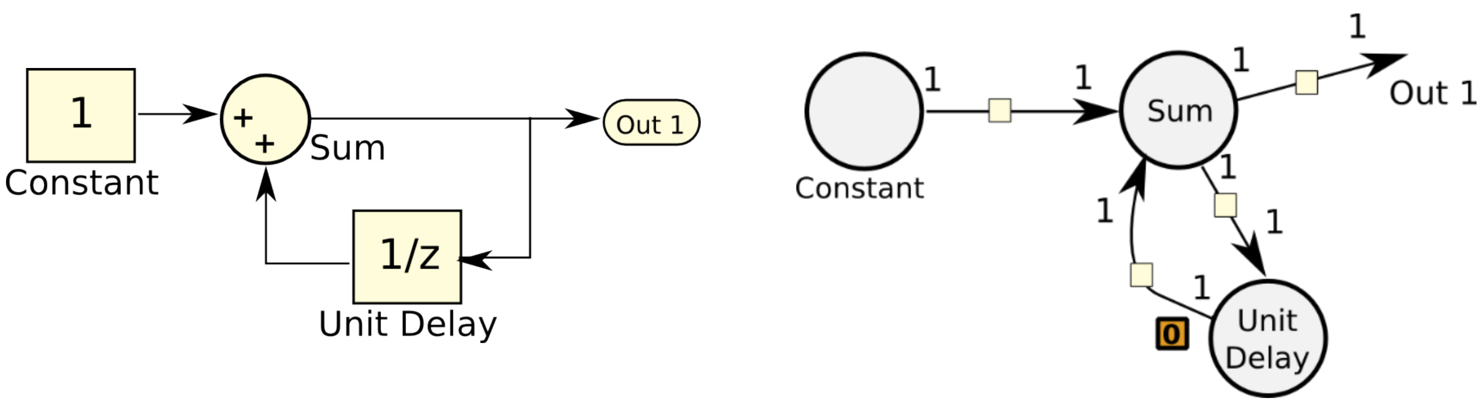

```
template <typename T, int SIZE>
Token<T,SIZE>::Token(){

}

template <typename T, int SIZE>
T &Token<T,SIZE>::operator[](int index)
{
   assert(0 <= index && index < SIZE);
   return Token<T,SIZE>::Data[index];
}
```

Apply Transformations (1)



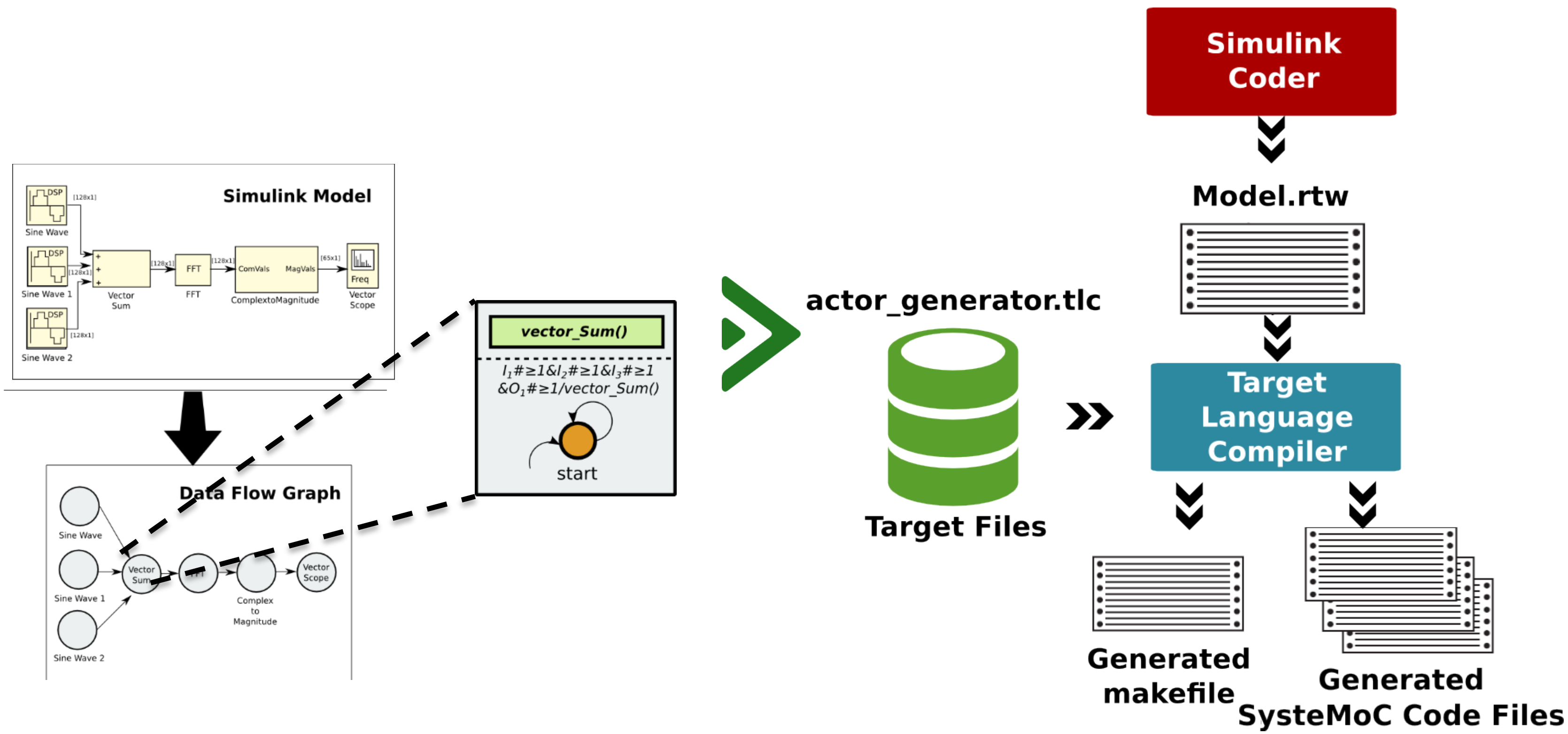
- Time-step execution allows multiple execution rates
- An actor and two FIFOs are added to regulate the write/read operations

Apply Transformations (2)

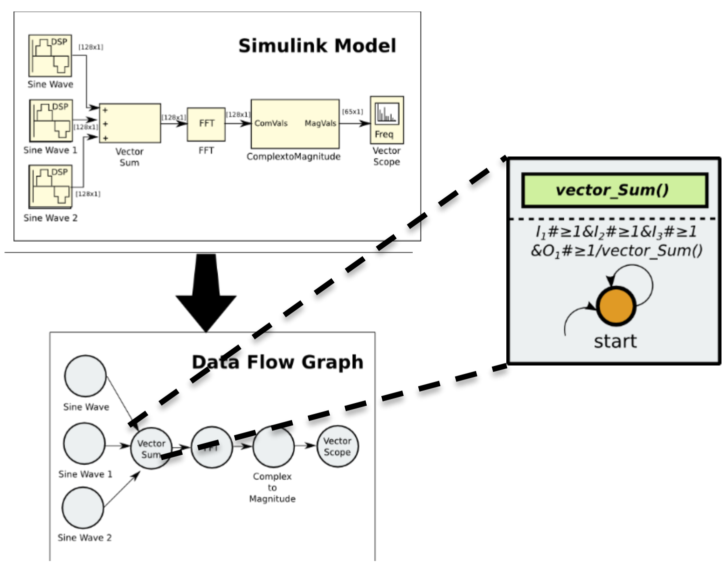


- Without a correct initialization dead-locks may be reached
- An initial token is added in the FIFO

SysteMoC Code Generation (1)

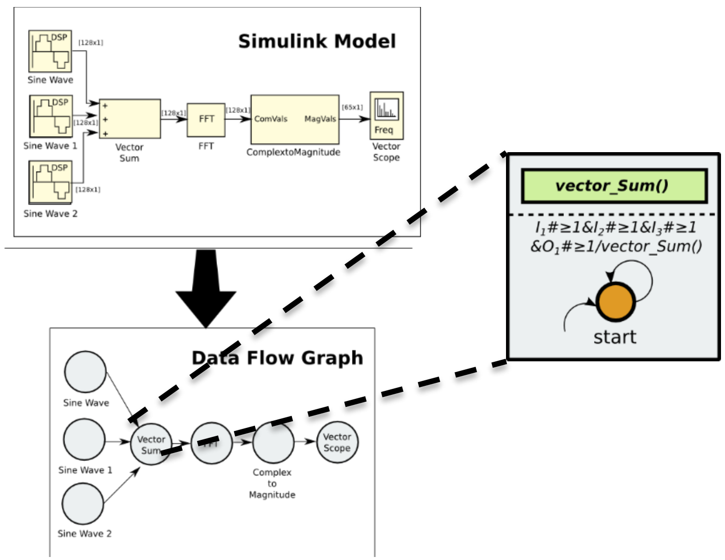


SysteMoC Code Generation (1)




```
class vector_sum: public smoc_actor
{
 public:
  /* Port Definition: */
  smoc_port_in<Token < real_T > > in_1;
  smoc_port_in<Token < real_T > > in_2;
  smoc_port_in<Token < real_T > > in_3;
  smoc_port_out<Token < real_T > > out_1;
  vector_sum(sc_module_name name): smoc_actor(name, start)
  {
    start = in_1(1)>> in_2(1)>> in_3(1)>> out_0(1) >>
          CALL(vector_sum::method_vector_sum) >> start;
  }
 protected:
  void method_vector_sum()
  {
    Token < real_T > token_out_0(128);
    vector_sum_initialize();
    for (int i = 0; i < 128; i++) {
      vector_sum_U.In1[i] = in_1[0].Data[i];
      vector_sum_U.In2[i] = in_2[0].Data[i];
      vector_sum_U.In3[i] = in_3[0].Data[i];
    }
   vector_sum_step();
   for (int i = 0; i < 128; i++)
    {token_out_1.Data[i] = vector_sum_Y.Out1[i];}
    out_0[0] = token_out_0;
  }
  smoc_firing_state start;  };
```

SysteMoC Code Generation (1)

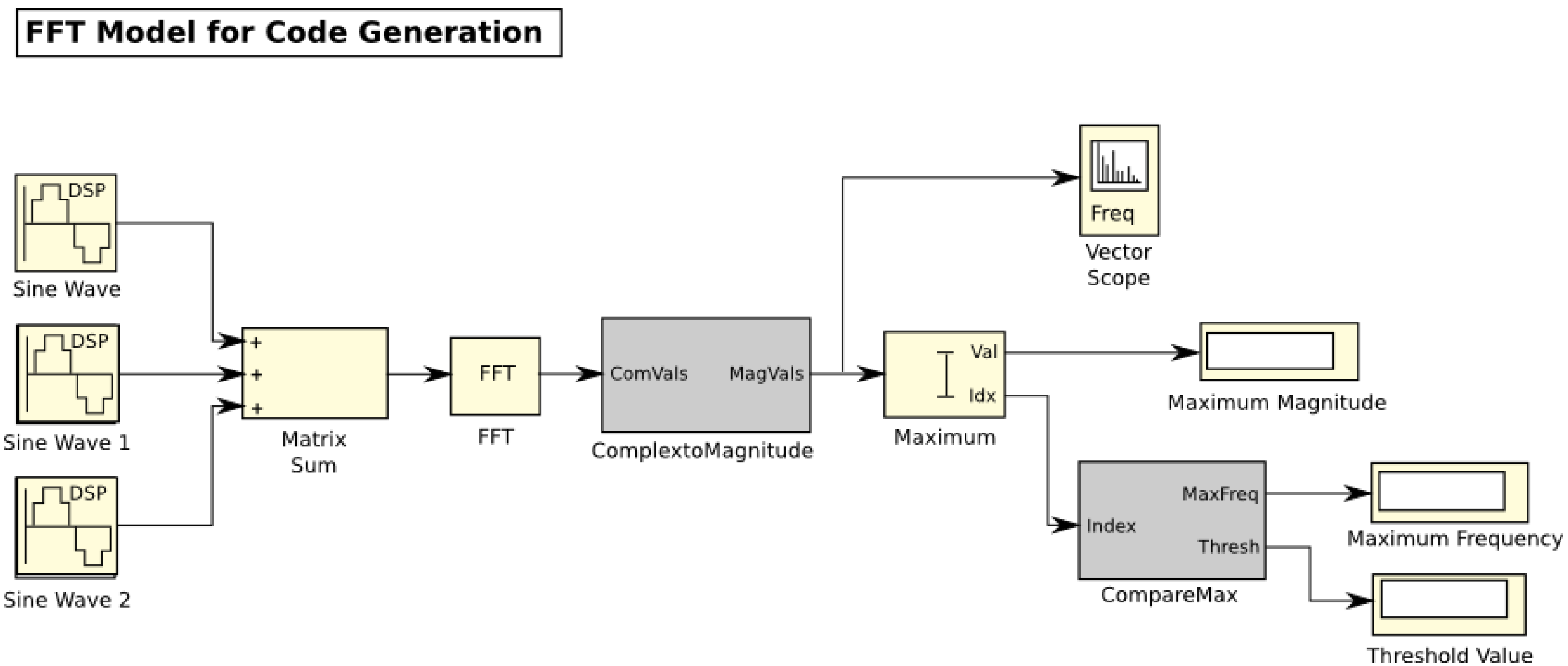



```
class Graph_System: public smoc_graph
{
 public:
  /* Actors Definition */
  sine_Wave sine_wave;
  sine_Wave1 sine_wave1;
  sine_Wave2 sine_wave2;
  vector_Sum vec_sum;
  FFT fft;
  ComplextoMag complex_to_mag;
  VectorScope vector_scope;


  Graph_System(sc_module_name name): smoc_graph(name),
{
    connectNodePorts(sine_wave.out,vec_sum.in_1);
    connectNodePorts(sine_wave1.out,vec_sum.in_2);
    connectNodePorts(sine_wave2.out,vec_sum.in_3);
    connectNodePorts(vec_sum.out,fft.in);
    connectNodePorts(fft.out,complex_to_mag.in);
    connectNodePorts(complex_to_mag.out,vector_scope.in);
  }
};
```

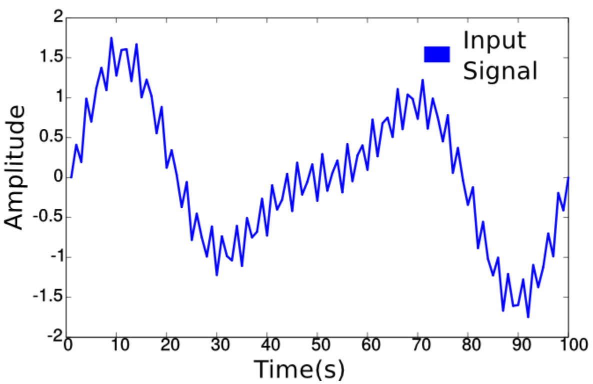
Case Study

Case Study (1)

Application that performs a FFT, extracts the maximum frequency and magnitude from an input signal

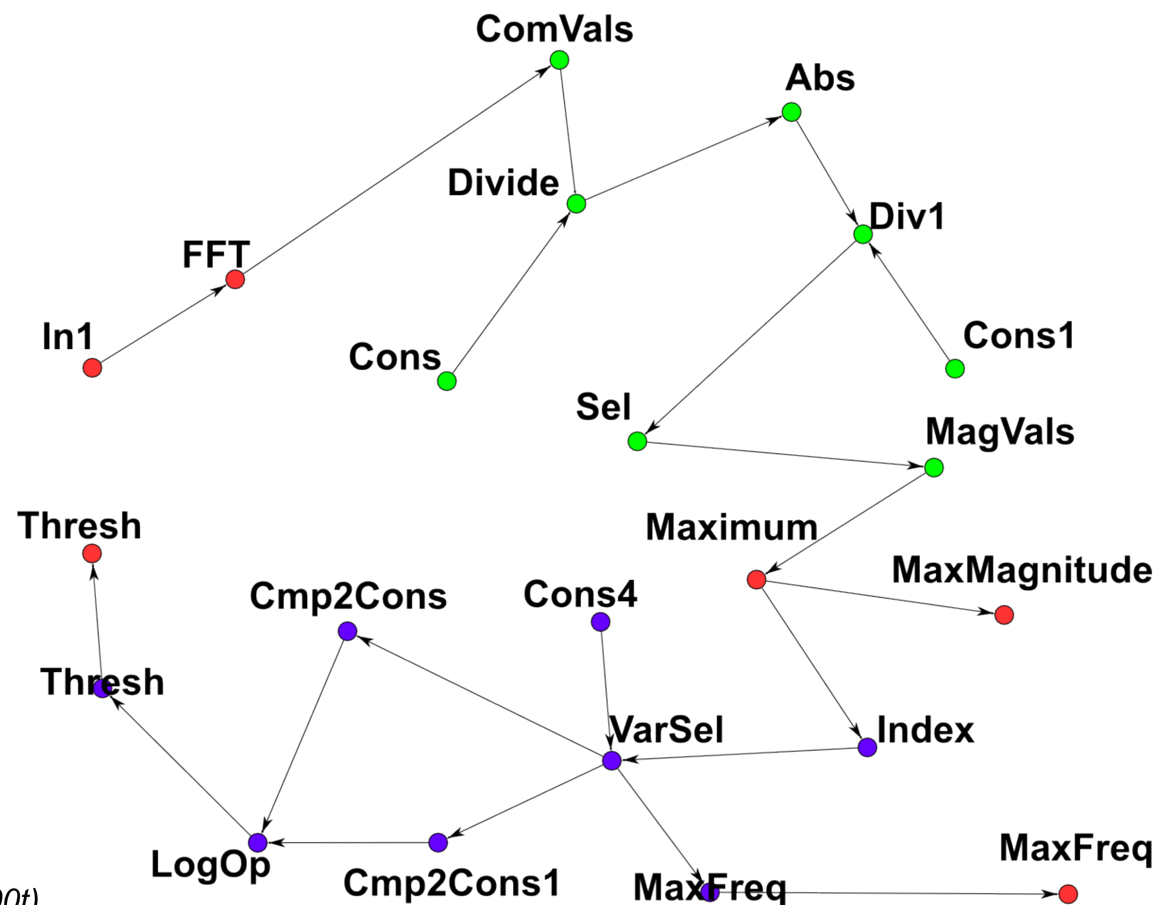


Case Study (2)



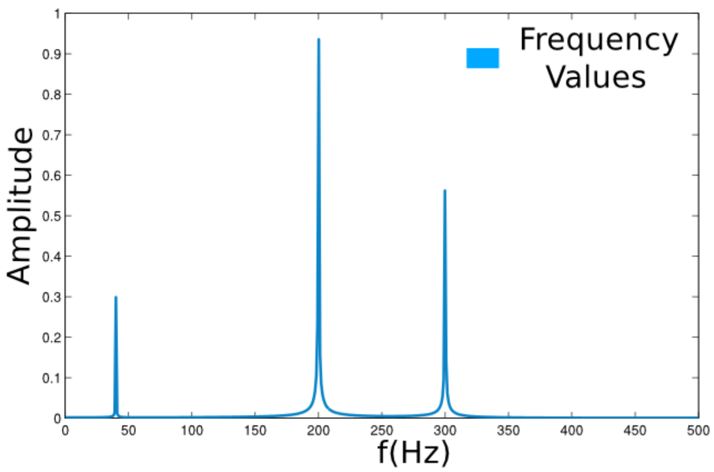
The input signal is the function

f(x) = 0.3sin(2π40t)+sin(2π200t)+0.6(2π300t)

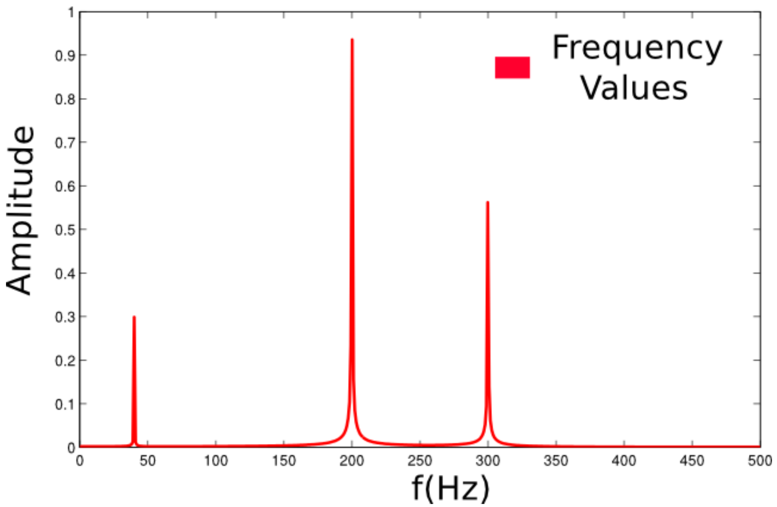


The DFG is composed by 21 actors and 16 FIFOs

Case Study (3)



Output of Simulink



Output of the SysteMoC actor network

Extract the three frequencies in the signal (40, 200 and 300 Hz.)

Comparison of related work

Work	MoC Employed	Library Required	Target Language	Multirate Systems	Triggered Systems	DSE	Offset handling
Zhou [3]	EFA	✓	Java	✓	✓	✗	✓
Warsitz [4]	KPN	✓	C and C++	✓	✗	✗	✗
Boström[5]	SDF	✓	Boogie	✓	✗	✗	✓
Chessa[6]	SDF	✓	C and C++	✗	✗	✗	✗
Caspi[8, 9]	SDF	✓	Lustre	✓	✗	✗	✓
Zhang[10]	SDF	✓	SysteMoC	✓	✗	✓	✓
Our work	**SDF**	✗	**SysteMoC**	✓	✓	✓	✗

Conclusions and Outlook

- We have presented a method for the automatic transformation of Simulink models to actor networks
- We are only constrained -and our translation coverage limited- by the class of Simulink blocks that can be converted by Simulink Coder™
- Independence from the input file (slx or mdl), and the ability of handling different data types (integers, floats, vectors as well as matrices)

Thank you

Questions?

References

[1] Keinert, J., Schlichter, T., Falk, J., Gladigau, J., Haubelt, C., Teich, J. U., & Meredith, M. (2009). SystemCoDesigner—an automatic ESL synthesis approach by design space exploration and behavioral synthesis for streaming applications. ACM Transactions on Design Automation of Electronic Systems (TODAES).

[2] Falk, J., Zebelein, C., Keinert, J., Haubelt, C., Teich, J., & Bhattacharyya, S. S. (2010). Analysis of SystemC actor networks for efficient synthesis. ACM Transactions on Embedded Computing Systems (TECS).

[3] Zhou, C., & Kumar, R. (2012). Semantic translation of simulink diagrams to input/output extended finite automata. Discrete Event Dynamic Systems.

[4] Warsitz, S., & Fakih, M. (2016). Simulink-Modell-Übersetzung in synchrone Datenflussgraphen. In MBMV (pp. 89-101).

[5] Boström, P., & Wiik, J. (2016). Contract-based verification of discrete-time multi-rate Simulink models. Software & Systems Modeling.

[6] Chessa, D. (2011). Conception and Implementation of Parallelism Analyses in MATLAB/SIMULINK Models for programming Embedded Multicore-Systems. Bachelorarbeit, Technische Universität München.

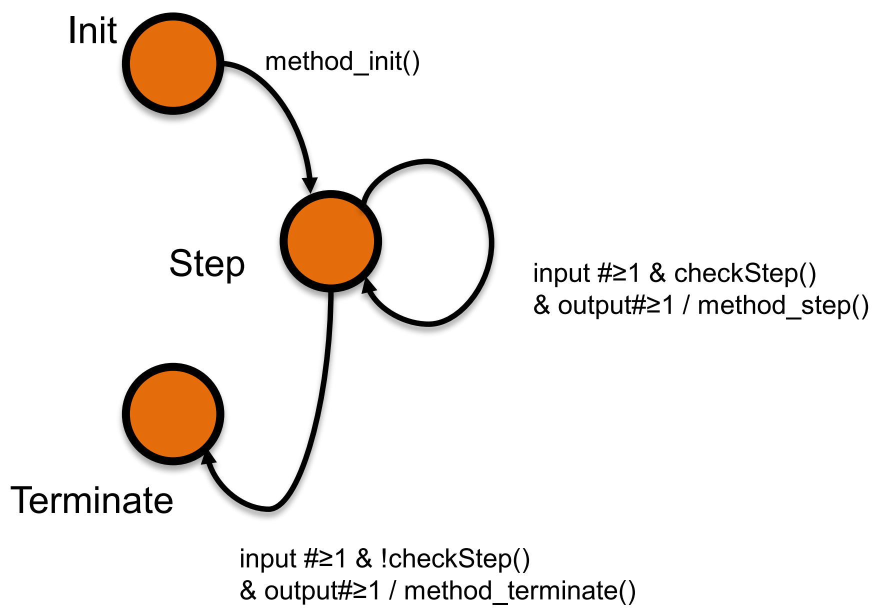
[7] Guesmi, K., & Hasnaoui, S. (2014, December). Translating of matlab/simulinlk model to synchronous dataflow graph for parallelism analysis and programming embedded multicore systems. In Design & Test Symposium (IDT), 2014 9th International (pp. 156-161). IEEE.

[8] Caspi, P., Curic, A., Maignan, A., Sofronis, C., Tripakis, S., & Niebert, P. (2003, June). From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications. In ACM Sigplan Notices (Vol. 38, No. 7, pp. 153-162). ACM.

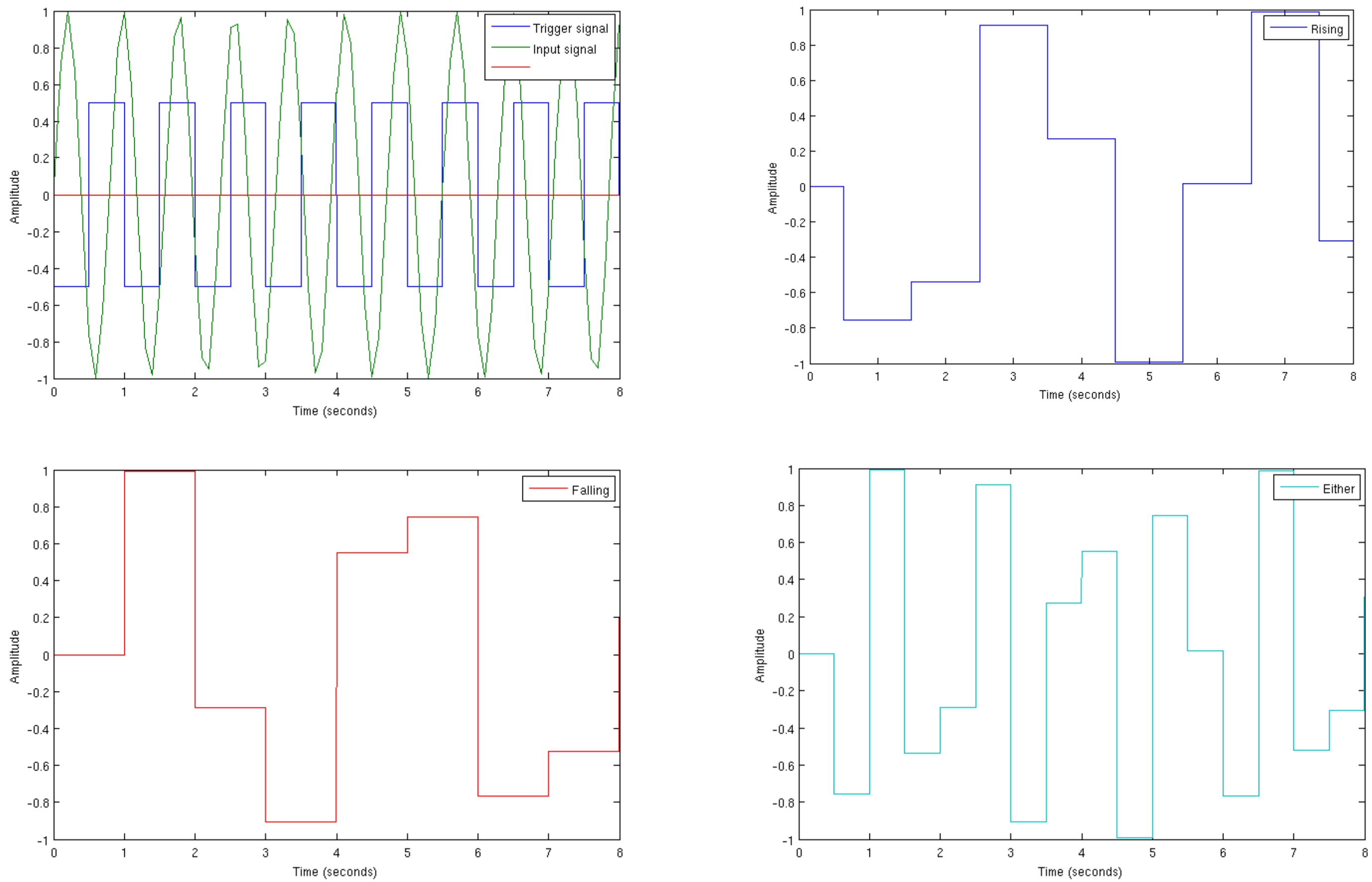
[9] Tripakis, S., Sofronis, C., Caspi, P., & Curic, A. (2005). Translating discrete-time Simulink to Lustre. ACM Transactions on Embedded Computing Systems (TECS).

[10] Zhang, L., Glaß, M., Ballmann, N., & Teich, J. (2015). Bridging algorithm and ESL design: Matlab/Simulink model transformation and validation. In Languages, Design Methods, and Tools for Electronic System Design (pp. 189-206). Springer International Publishing.

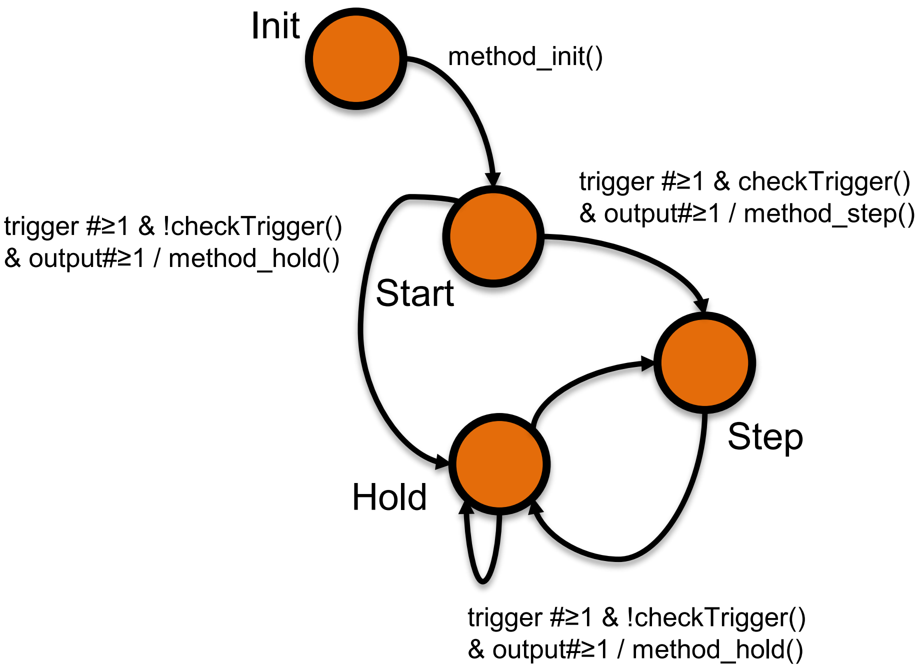
Finite State Machine for Actors



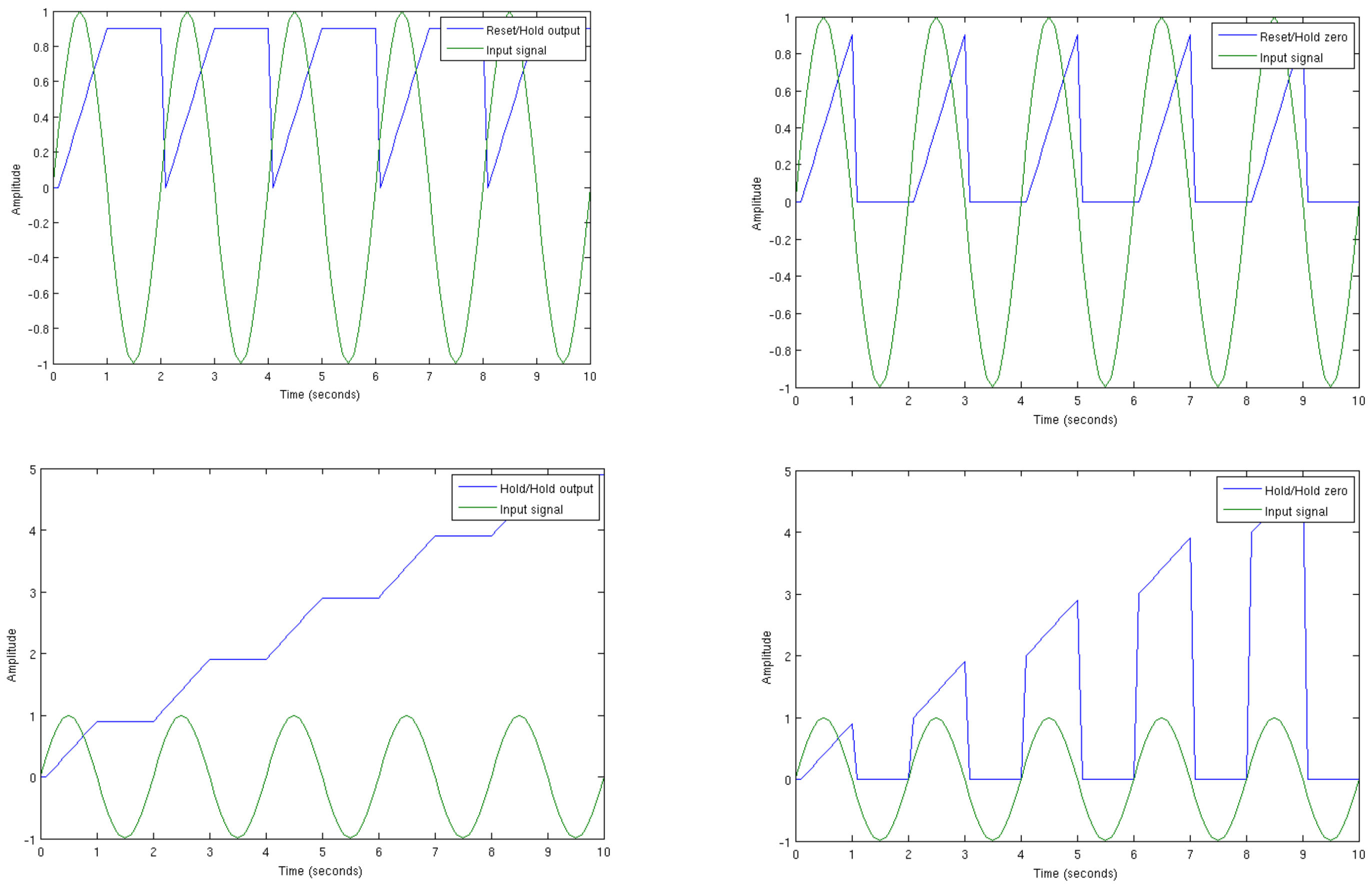
Triggered Subsystems



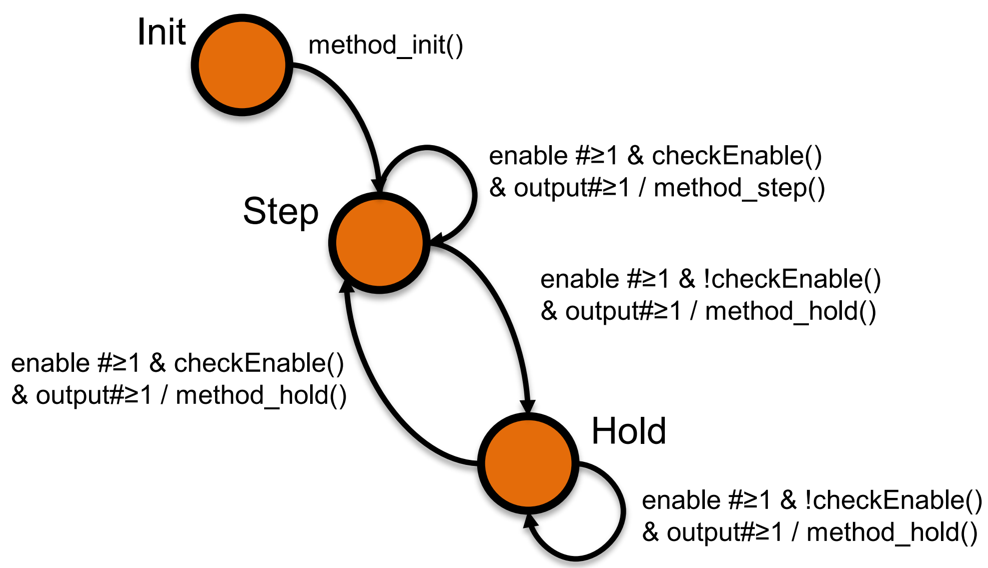
Finite State Machines for Triggered Subsystems



Enabled Subsystems



Finite State Machines for Enabled Subsystems



Finite State machine for hold/hold output