Modelica library for logic control systems written in the FBD language

Alberto Leva, Filippo Donida, Marco Bonvini*, Lorenzo Ravelli*
Dipartimento di Elettronica e Informazione, Politecnico di Milano
Via Ponzio, 34/5 – 20133 Milano, Italy
{donida,leva}@elet,polimi.it
*former student at the Politecnico di Milano

Abstract

The paper describes a Modelica library for the simulation of logic control systems written in the FBD (Functional Block Diagram) language as defined in the IEC61131.3 standard. The library contains not only strictly logic blocks, but also the main types of industrial PID controllers. Models of different complexity levels are included, so that the user can specify a control system as a continuous-time model (for fast simulation to check whether or not a control strategy solves the problem at hand) or an event-based one (for precise evaluation of the algorithms' behaviour).

1. Introduction

In many control domains, particularly – but not exclusively – in the process control field, a correct representation of the control system connected to the plant being investigated is of paramount importance [7, 1, 3, 18, 19, 8, 9, 4, 11, 13, 17, 21, 15]. In many cases, the structuring and the subsequent tuning of that control system is even the main goal of the simulation activity; and also if control commissioning is not the primary purpose of the simulation, having a correct and realistic control representation is always important in order to draw meaningful conclusions.

Nowadays, more and more control systems are implemented adhering to the IEC61131.3 standard [1, 2, 6, 7, 20, 10, 11, 14, 16], that defines five programming languages (Ladder Diagram or LD, Sequential Functional Chart or SFC, Functional Block Diagram or FBD, Structured Text or ST, Instruction List or LD) basically oriented to logic control, although most systems adhering to the standard also offer modulating control functions. In the last years, the IEC61131.3 standard has become very popular in the arena of PLC programming, therefore spreading out in a vast number of contexts and applications [2, 10, 8, 16, 15, 5].

As such, having the IEC61131.3 standard available in the Modelica environment is of great help, for at least two reasons. First, if an *industry* standard is *uniformly* adopted, there is (ideally) no room for ambiguities in the communication between the people who own and/or run the plant, and the analysts who create the simulator and realise the necessary studies (some issues may still arise owing to the fact that virtually every standard is the result of a compromise, and therefore very frequently exists also in the form of so-called “dialects”, but addressing that problem is apparently beyond the scope of this research). Second, the solutions found at the simulation level are deployed to the target control architecture in a very straightforward way.

For the reasons above, a free (GPL) FBD Modelica library is being developed at the Politecnico di Milano. The present state of that library is described in this paper, that is organised as follows. Section 2 describes the organisation of the library, briefly list its contents, and presents some selected blocks with a minimum of detail. Section 3 discusses two examples. The first aims at showing the importance of having the regulators described both at a simplified (continuous-time) and at a detailed (event-based) level. The second shows some library blocks applied to the control of a small manufacturing system, to illustrate how the obtained Modelica schemes are easily understood by people developing code for the typical industrial control architectures. Finally, section 4 reports some conclusions, and the future plans of the research.

2 Library organisation

The library comes in a single Modelica package named *FBD*, and organised in subpackages as sketched below:

- the *FBD.OneBitOperation* subpackage implements basic logical operations,
- the *FBD.CompareOperation* subpackage implements comparisons (the <, >, >= ,<= ,= operators) on the Integer and Real types,
- the *FBD.Counter subpackage* provides up/down counters,
- the *FBD.MathOperation* subpackage implements the basic mathematical instructions,
- the *FBD.Timer* subpackage provides timers (and is similar to the Counter one),
- the *FBD.NBitOperation* subpackage implements logical operation on arrays of bits,
- the *FBD.LinearSystems* subpackage provides linear, time invariant dynamic systems in the continuous and discrete time, as typically specified in IEC-compliant control code development environments,
- the *FBD.IndustrialController* subpackage contains several industrial controllers, including of course several types of PID,
- the *FBD.Test* subpackage contains test simulators for each FBD block, individually, to allow for a precise comprehension of its functionalities,
- and finally the *FBD.Applications* subpackage provides some examples of use of the FBD blocks of the library.

For obvious space reasons we do not describe the blocks here, referring the interested reader to the library documentation. A couple of remarks are however worth some lines.

First, for *every* component a “test” model is provided, to allow the user to fully understand how that component works, and possibly disambiguate situations where the available specifications are not fully univocal; everyone wishing to extend the library (contributions are of course welcome in the GPL spirit) is strongly encouraged to do the same.

Second, especially for regulators, both continuous-time and event-based models are present. The former type of model allows for faster simulation, and is the choice of election when the purpose is to check the correctness of a control *strategy*. The latter is apparently less time-efficient, but allows to check the behaviour of a control *algorithms*. The library therefore allows to perform both types of simulation, and even to mix the two, e.g. by convenient use of model replaceability, and top-level variables. To limit the performance loss, equations (not algorithms) were used in event-based models, so as to allow those models to be manipulated with the rest of the simulator. Doing so involves some limitations when porting a pre-existing algorithm into the library, since for example multiple assignments are not allowed. It is the authors' opinion, however, that an accurate translation in the form adopted by the presented library is possible for of any control algorithm one may come across.

For example, the following Modelica code is the event-based implementation of an ISA PID with antiwindup, manual and tracking modes, and bumpless mode switch [17, 18].

```
function Der "Thi sfunction represents a derivative action"
  input Real sp;
  input Real pv;
  input Real pv_old;
  input Real Td;
  input Real Ts;
  input Real N;
  input Real d_old;
  output Real d;
algorithm
  d := Td/(Td + N*Ts) * d_old - Td*N/(Td + N*Ts) * (pv - pv_old);
end Der;

model Proportional
  RealInput sp "set point";
  RealInput pv "process variable";
  RealOutput p "control signal";
  parameter Real Ts =  0.1 "sample time [s]";
  parameter Real K =   5 "proportional constant";
  parameter Real b =   1 "set point weight";
protected
  discrete Real sp_d;
  discrete Real pv_d;
  discrete Real p_d(start=0);
equation
  when sample(0,Ts) then
    sp_d = sp;
    pv_d = pv;
    p_d  = p;
    p = Pr(pre(sp),pre(pv),K,b);
  end when;
end Proportional;

model Integral
  RealInput sp "set point";
  RealInput pv "process variable";
  RealOutput i "control signal";
  parameter Real Ts =  0.1 "sample time [s]";
  parameter Real Ti =  5 "integral time";
protected
  discrete Real i_d(  start=0);
  discrete Real sp_d(  start=0);
  discrete Real pv_d(  start=0);
equation
  i_d = i;
  when sample(0,Ts) then
    sp_d = sp;
    pv_d = pv;
    i_d = Int(pre(sp),pre(pv),Ti,Ts,pre(i_d));
  end when;
end Integral;

model Derivative
  RealInput sp "set point";
  RealInput pv "process variable";
  RealOutput d "control signal";
  parameter Real Ts =  0.1 "sample time [s]";
  parameter Real Td =  5 "derivative time";
  parameter Real N =  10 "derivative filter";
protected
  discrete Real d_d(  start=0);
  discrete Real d_d2(  start=0);
  discrete Real pv_d(  start=0);
  discrete Real pv_d2(  start=0);
  discrete Real sp_d(  start=0);
equation
  d_d  = d;
  when sample(0,Ts) then
    pv_d  = pv;
    sp_d  = sp;
    d_d2  = pre(d_d);
    pv_d2 = pre(pv_d);
d_d= Der(pre(sp_d),pre(pv_d),pre(pv_d2),Td,Ts,N,pre(d_d2));
  end when;
end Derivative;

model PID_parallel_AW_Tr_AutoMan
  RealInput sp "set point";
  RealInput pv "process variable";
  RealInput tr "signal followed during the tracking mode";
  RealInput CSman "control signal for manual mode";
  BooleanInput TS "flag for the tracking mode";
  BooleanInput MAN "flag for the manual mode";
  RealOutput cs "control signal";
  Proportional P( Ts=Ts,K=K,b=b) "Proportional block";
  Derivative D(    Ts=Ts,Td=Td,N=N) "Derivative block";
  Integral I(     Ts=Ts,Ti=Ti) "Integral block";
  parameter Real Ts = 1 "sample time [s]";
  parameter Real Ti = 8 "integral time";
  parameter Real Td = 5 "derivative time";
  parameter Real K = 10 "proportional constant";
  parameter Real b = 1 "weight of the set point in the P action";
  parameter Real N = 10 "derivative filter";
  parameter Real CSmax = 1 "Max cs value";
  parameter Real CSmin = 0 "min Cs value";
protected
  Real control;
equation
  P.sp = sp;
  D.sp = sp;
  I.sp = sp;
  P.pv = pv;
  D.pv = pv;
  I.pv = pv;
  control = if (MAN==false)
            then I.i + P.p + D.d
            else CSman;
  cs = if (TS==true and MAN==false)
       then tr
       else max(CSmin,min(CSmax,control));
end PID_parallel_AW_Tr_AutoMan;
```

3 Examples

We now report two simulation examples. the first is aimed at showing the usefulness of the possibility of simulating the same regulator as continuous-time and as event-based model, while the second shows a “small but realistic” application of the presented library.

3.1 Example 1

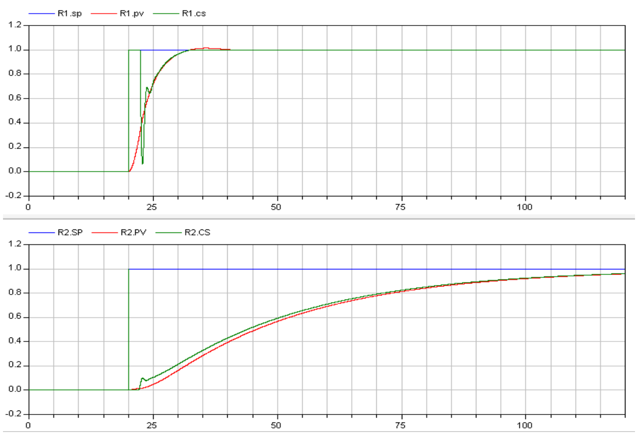
This example refers to some PI/PID control loops, and deals with set point step and ramp responses where the antiwindup mechanism of the regulator comes into play. The process to be controlled is described by the transfer function

$$P(s)=\frac{1}{1+2s+s^2/0.016}$$

and the PID regulator

$$R(s)=10\left(1+\frac{1}{30s}+\frac{3s}{1+0.3s}\right)$$

is applied to it, in the continuous-time version and as an event-based model with a sampling time of 0.01 s.



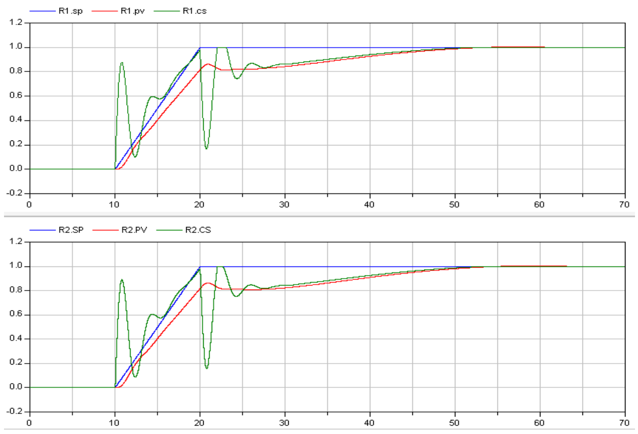


Figure 1: results of example 1.

Figure 1 above shows the comparison between the continuous-time (R1) and event-based (R2) controller implementation in the case of a ramp response (left column of plots) and of a step response (right column): SP, PV and CS stand for Set Point, Process (controlled) Variable, and Control Signal, respectively. Apparently, simulating the same controller as a continuous-time or an event-based model (i.e., as it will really be implemented) can give very different results, depending not only on the controller parametrisation, the sampling time and other very well known facts, but also on the control law being incremental or positional, of the antiwindup type, and so on (facts that conversely are frequently overlooked). The example therefore backs up the usefulness of the presented library as far as the control behaviour evaluation is concerned.

3.2 Example 2

This example shows the control of a small manufacturing system where parts are fed to the working area by a conveyor, machined, and then taken away by another conveyor. The detailed sequence of operations is as follows:

- lead one part near the machining area entrance with an input belt,
- push the part into the machining area with an input piston,
- machine the part (drill a hole with a controlled-speed machining head)
- push the part out of the machining area with an output piston,
- and finally lead the part away with an output belt.

The considered machine is synthetically described in figure 2

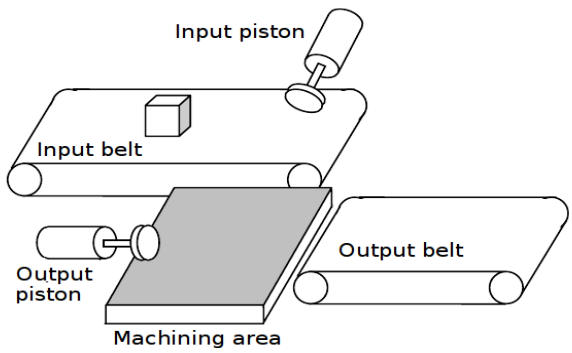


Figure 2: schematic drawing of the machine considered in example 2.

Figure 3 shows the Modelica scheme using some library blocks (mostly set point generators, PIDs, and logic elements), while a sample of simulated transients is given in figure 4.

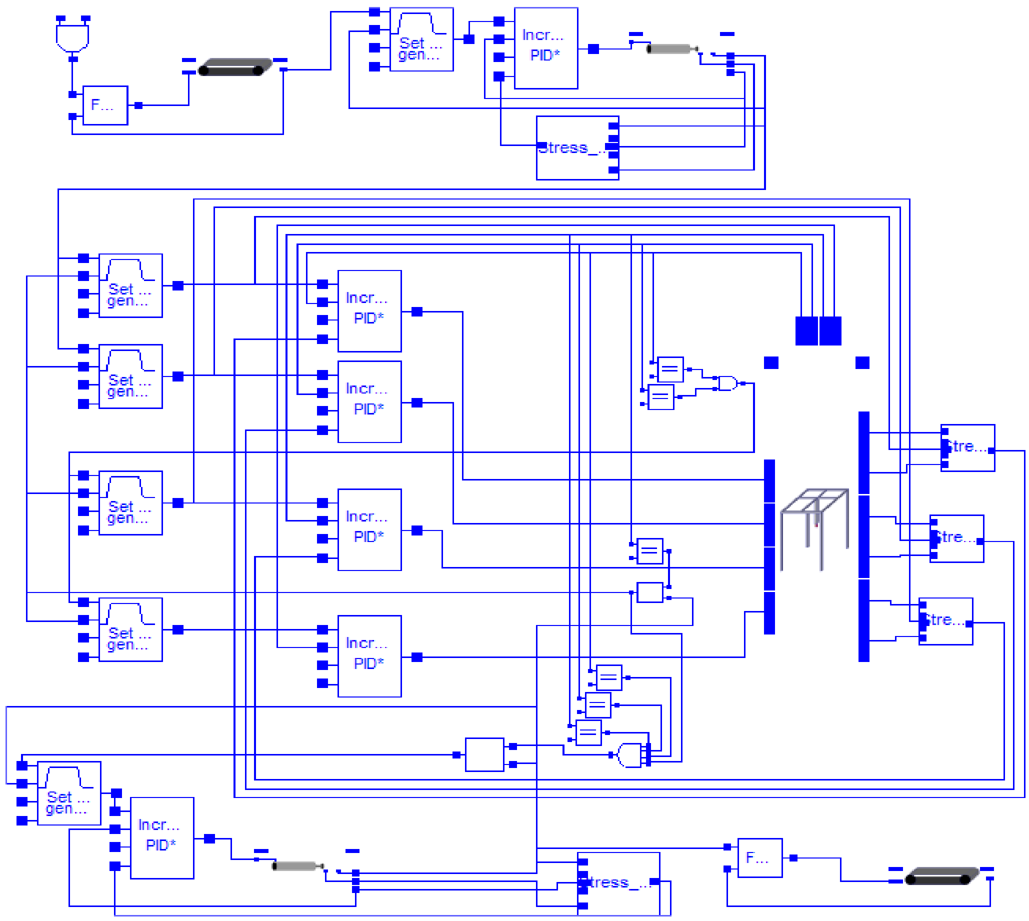


Figure 3: the Modelica scheme using the presented FBD library used in example 2.

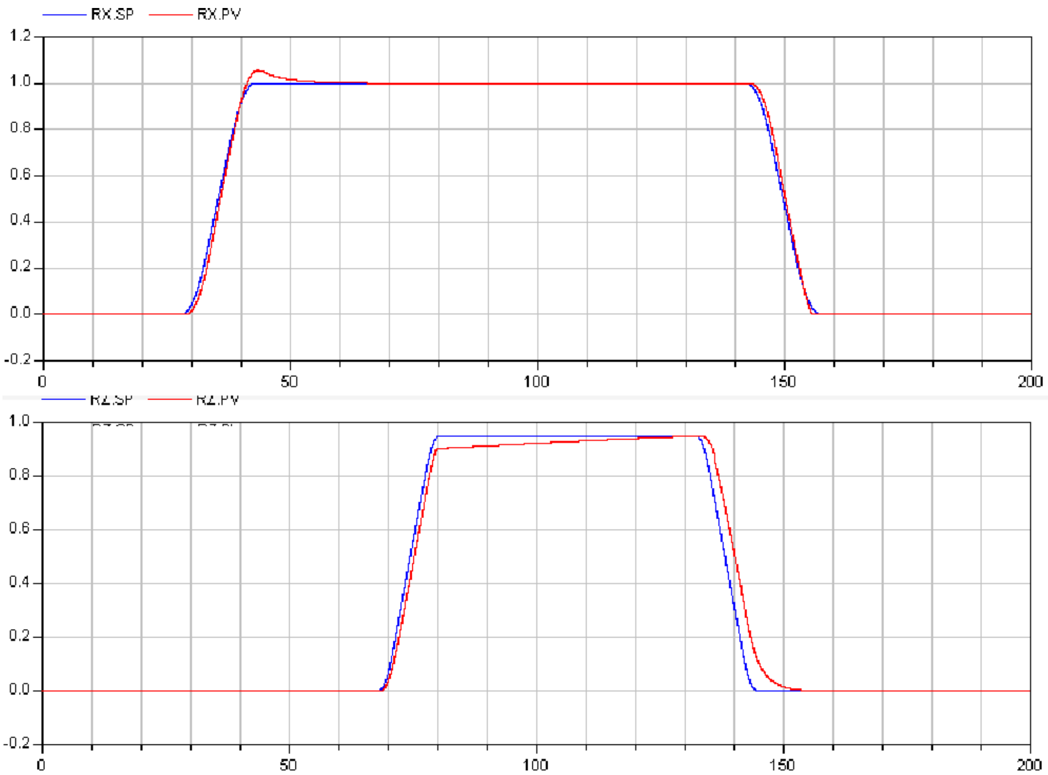


Figure 4: some simulated transients referring to example 2; the upper plot shows the drilling head x position (red) and set point (blue), the lower plot shows the drilling depth (red) and set point (blue).

The similarity of figure 3 with the schemes encountered in many control code development systems are apparent. The example therefore backs up the usefulness of the presented library as far as the clarity of the control specification (in terms of a widely accepted industrial standard) is concerned.

4 Conclusions

A free (GPL) Modelica library for the simulation of logic control systems written in the FBD (Functional Block Diagram) language was presented.

The library adheres to the FBD specifications as defined in the IEC61131.3 standard, and contains not only strictly logic blocks, but also the main types of industrial controllers, particularly of the PID type. The adoption of an industrial standard facilitates information sharing and greatly reduces ambiguities.

With the presented library, that the user can specify a control system as a continuous-time or an event-based model , for maximum flexibility in fulfilling the simulation needs.

Some simulations were presented to illustrate the usefulness of the library, which will be extended in the future, with respect to both FBD and other IEC-compliant languages.

References

[1] T. Sato, E. Yoshida, Y. Kakebayashi, J. Asakura, N. Komoda, Application of IEC61131-3 For Semiconductor Processing Equipment, Emerging Technologies and Factory Automation. Proceedings. 2001 8th IEEE International Conference on, 2001.

[2] J. Huang, Y. Li, W. Luo, X. Liu, K. Nan, The Design of New-Type PLC based on IEC61131-3, Proceeding of the Second Internadonal Conference on Machine Learning and Cybernetics, Xi, 2-5, November 2003.

[3] D. E. Rivera, M. Morari, and S. Skogestad, Internal model control 4. pid controller design, Ind. Eng. Chem. Res., vol. 25, pp. 252–265, 1986.

[4] H. Takada, H. Nakata, S. Horiike, A Reusable Object Model for Integrating Design Phases of Plant Systems Engineering, Proceedings of the Fourth International Conference on Computer and Information Technology (CIT'04).

[5] H. Taruishil, S. Kajiharal, J. Kawamotol, M. Ono, H. Ohtani, Development of Industrial Control Programming Environment Enhanced by Extensible Graphic Symbols, SICE-ICASE International Joint Conference 2006 in Bexco, Busan, Korea, Oct. 18-2 1, 2006.

[6] Y. Qiliang, X. Jianchun, W. Ping, Water Level Control of Boiler Drum Using One IEC61131-3-Based DCS, Proceedings of the 26th Chinese Control Conference, Zhangjiajie, Hunan, China, July 26-31, 2007.

[7] M. Bonfe', C. Fantuzzi, L. Poretti, PLC Object-oriented programming using IEC61131-3 norm languages: an application to manufacture machinery, in Proc. of IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics, vol. 2, pp. 787-792, 2001.

[8] [Online]. Available http://www.plcopen.org.

[9] J. Roger Folch, J. Pérez, M. Pineda, R. Puche, Graphical Development of Software for Programmable Logic Controllers, 12th International Power Electronics and Motion Control Conference.

[10] [Misc]. DeltaV: Monitor and control software.

[11] [Misc]. Labview: http://www.ni.com/labview.

[12] A. Nobuo, I. Kenichi, Y. Eiji, Application portfolios for stardom, 12th International Power Electronics and Motion Control Conference.

[13] M. Otter, K. E. Årzén, I. Dressler, StateGraph-A Modelica Library for Hierarchical State Machines, 4th International Modelica Conference, March 7-8, 2005.

[14] O. Johansson, A. Pop, P. Fritzson, Engineering Design Tool Standards and Interfacing Possibilities to Modelica Simulation Tools, 5th International Modelica Conference, September 4-5, 2006.

[15] E. Tisserant, L. Bessard, M. de Sousa, An Open Source IEC 61131-3 Integrated Development Environment, Industrial Informatics, 5th IEEE International Conference on, 2007.

[16] [Online]. ISaGRAF: http://www.icpdas.com/products/PAC/i-8000/isagraf.htm

[17] [book]. O'Dwyer Aidan, Handbook of PI and PID Controller Tuning Rules, Imperial College Press.

[18] [book]. K. J. Åström and T. Hägglund, Advanced PID control, ISA - The Instrumentation, Systems, and Automation Society, 2005.

[19] [book]. K. J. Åström and T. Hagglund, PID Control Theory, Design and tuning, ISA, 1995.

[20] L. Desbourough, R. Miller, Increasing customer value of industrial control performance monitoring – Honeywell's experience, Sixth International Conference on Chemical Process Control, AIChE Symposium Series Number 326 (Volume 98), 2002.

[21] O. Johansson, A. Pop, P. Fritzson, A functionality Coverage Analysis of Industrially used Ontology Languages, in Model Driven Architec-ture: Foundations and Applications (MDAFA), 2004, 10-11 June, 2004, Linköping, Sweden.