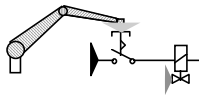


Automata in industrial Control

The Sequential Function Chart

Introduction

General

In control engineering there is a need for formal representations of control sequences which satisfy the following requirements: i) it should be suitable as design representation, enabling structured problem solving and communication within a team, ii) it should powerful enough to tackle the complexity of industrial control problems including error handling and interaction with a physical plant and iii) it must be complete and unambiguous. As a consequence the representation must be transparent in the sense that for every situation, it is obvious, i.e. graspable within less than a minute, what the controller should be doing and will be doing next. Textual specification and documentation of control sequences is a cumbersome task and is only suitable for very simple linear structures. Much more powerful are graphical representations of automata, e.g. state machine (Moore ore Mealy), flow chart with extensions, Petri Net or the Sequential Function Chart (SFC). Since the SFC meets the transparency requirements mentioned above, is equivalent to Petri Nets and a generalization of state machines and is also a formal programming language in the IEC61131-3 standard, it will be used in the sequel.

In the PLC-programming community there are many reservations and also prejudices about using SFC. One reason might be that automata theory is taught at a very abstract and formal level. Quintuple of sets define automata and an ugly formal language is used to state theorems. Here, automata will be introduced in a pragmatic way aimed to solve industrial control problems. First, it is shown, how SFC has to be formally drawn and how they can be used to solve typical control problems. Since SFC is equivalent to Petri Net, the entire theory of Petri-Net analysis is also applicable for SFC. Analysis of SFC can reveal several structural deficiencies which can result from a hurried design. In the second part, basics of SFC-analysis are explained with the aim, that the reader is able to understand the error messages, which might be created by an SFC-compiler and is able to correct the SFC.

Where to use SFC

'Can you program a PID-controller in a SFC?' This question doesn't attest deep knowledge about how to use SFC but rather reflects reservations against using SFC. Nevertheless there has to be an answer to the question, where to best use SFC. All control tasks can roughly be divided into continual and event-driven tasks. This is shown in Figure. Continual tasks are incessantly executed. This could be recording process values, checking the violation of limits or doing feedback control. The best implementation of these tasks is done in analog circuitry; in PLC this behavior is approximated with periodical calculation. Limit switches and user interaction create events that have to be processed within some logic. As a result they might turn actuators on or off. Most likely this logic is not just a memory less function, but depends on the state of several state variables. As soon as this logic is more complex, i.e. depending on more than two state variables, SFC offers an optimal solution.

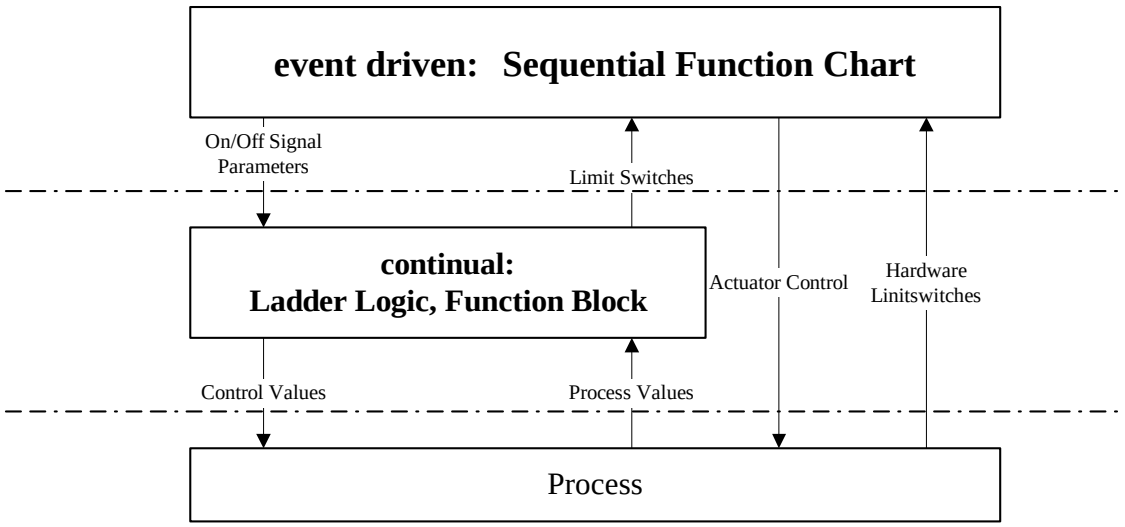


Figure 1: Continual and event driven control logic

Why to use SFC

As shown in the past, state dependent logic can also be implemented in Ladder Logic or with Function Block. SFC offers a large set of advantages which is yet only partially explored in industry. There are several very appealing advantages in using SFC:

- a SFC is transparent in the sense, that for every situation your are immediately able to see, what the controller is doing now and what it will be doing next

- Software design is always complete including flexible error handling, since normal operation and error handling are visually combined

- Plant startup is done in an extremely short time:

  - Software errors can easily be found. There are no highly challenging debug problems to solve in a stressing environment.

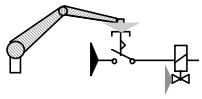
  - Since SFC allows local modification of a sequence without changing the behavior of the remaining program, errors can be corrected without creating several new errors. Logic with several state variables is extremely difficult to modify without creating side effects.

  - There is a systematic way to setup the test procedure for startup of SFC-controlled plants. This will become evident when the reader is familiar with SFC

- The SFC-graph with lists of actions and transitions, together with an interface description of the module is complete and very comfortable software documentation

- Since SFC is very simple to read, it can be used to define software requirements. Using SFC Operators are able to make valuable contributions when software requirements are defined in detail. The immediate consequence is that the operators wish to have an animated SFC on the HMI.

- SFC is also a suitable representation to specify a production recipe.



As explained in the previous section, a SFC is equivalent to a Petri-Net. Possible advantages of SFC are, that control actions and go-on conditions are uniquely represented with one type of element, whereas in Petri-Net this not the case. As a consequence, SFC is easier to implement.

Application areas

SFC and more generally automata have a wide field of application. The following is not complete; it should document its wide field of application.

Information Sciences:

- Software Design (State machine in UML)
- Modeling, analysis and programming of parallel/concurrent processes
    - shared resources
    - Data base access
    - Real-time systems
- Modeling, simulation and analysis of queues
- Many application in the area of communication

Plant control

- Modeling, analysis and programming of plant control software

Production:

- Workflow specification, analysis and simulation
- Modeling, simulation and analysis of production lines