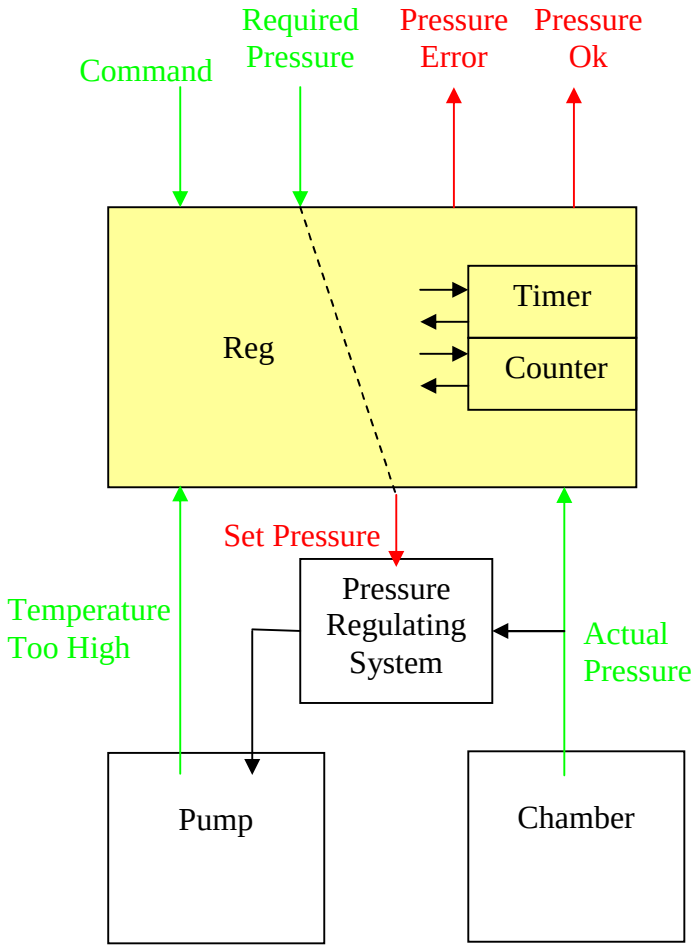
StateWORKS in industrial control

Introduction

StateWORKS covers both the design and implementation of a control system. StateWORKS is a universal system; it can be used for creating any software system. Anyway, there are essential differences between for instance an industrial control and a telecommunication system. Particularly, the telecommunication system does not “know” digital inputs or DA- and AD-converters which are the essence of the industrial control.

StateWORKS RTDB contains a large set of object types which allows for handling of frequently used input / output signals. This case study shows the design process of such a system including the discussion of the basic Real Time Data Base (RTDB) objects required for industrial control. RTDB is the heart of the StateWORKS run-time systems and is specified in StateWORKS Studio during the design process.

Example



Inputs:

- Command (Number – integer, 1: Start)
- Temperature Too High (Bool, HIGH: Temperature too high)

- Actual Pressure (float, value from 12-bit AD converter: 0…4095 corresponding to 0...1000 mBar)
- Required Pressure parameter (float, value from operator: 0…1000 mBar).

Outputs:

- Alarm (String, Message: Pressure error)
- Digital Signal (Bool, HIGH: Pressure is Ok, to control LED)
- Set Pressure (float, Required Pressure parameter: 0…1000 mBar corresponding to 0…2047 for 11-bit DA converter).

Internal (at least):

- Timer
- Counter.

Requirements:

The discussed system is a typical system where a regulator does the principal job stabilizing some value (to make it more realistic we called this value a Pressure). This regulator is supervised by a control system which task is to supply the settings for the regulator and to detect malfunctioning. In addition, the supervising system is the interface to the operator, informing him about the control situation and passing commands or required values to the control system.

The Reg control system is to supervise an automatic control system (Pressure Regulating System) which stabilizes a Pressure in a chamber. The Regulator receives the Actual Pressure value and controls the Pump.

The operator defines the Set Pressure value by setting the Required Pressure value and starts the Reg supervision sending the Command Start (number = 1).

If the Pressure value reaches the required value ±5% of the Required Pressure it is signalled by a LED controlled by a Pressure Ok signal. If the Required Pressure value is not reached in a certain time the control system issues an alarm message to inform the operator about the failure and stops the supervision.

If the Required Pressure value is once reached and later exceeds the allowed range the control system should tolerate it, if it does not last longer than a certain time and if it occurs not more than a certain number of times. Otherwise the system should switch off the supervision and produce the alarm informing the operator about the failure. The system restarts the supervision on receiving again the command Start.

If the operator changes the Required Pressure value the Timer as well as the counting of failures should be reset and begin from 0.

The control problem

The control system as described above has to operate in a heterogeneous environment. It receives different type inputs: an integer number (Command), two float numbers (Required Pressure and Actual Pressure) and a Boolean signal (Temperature Too High). In addition, the Timer and Counter will generate expiration signals (OVER). On the outputs side, the control system produces also signals of different types: a float number (Set Pressure), alarm text (Pressure Error) and Boolean signal (Pressure Ok). In addition, the Timer and Counter are controlled by commands, like Start, Reset or Stop.

The input and output signals have several properties. The signal Actual Pressure coming from an 12-bit AD-converter is a number in the range 0…4095 which corresponds to 0…1000 mBar but the range which is considered as correct is ±5% of the Required Pressure. The practical version could be more complex: the range limits could be parameters or x% of the Required Pressure where x is a parameter. The imaginable variation of signal properties is large. Similarly, we could analyze all signals showing their varieties. In system design we have also to foresee possibilities to extend the requirements in the future.

The Set Pressure output value is a number in the range 0…1000 mBar and it is passed to an 11-bit DA-converter which output changes than in the range 0...2047.

SWLab simulates 12-bit DA- and AD-converters for AD- and DA-converters. Therefore, we will use the entire range for AD-converter (0...4095) but only half of the range for the DA-converter, moving its range by 2048 (offset).

The interesting question is how to master the control, how to filter the essential control factors from the irrelevant details. The problem seems at first sight to be relatively simple but if we consider all details it is getting complex.

The StateWORKS solution

Behavior

StateWORKS is based on a strict partition between the data and control flow. Using StateWORKS approach we concentrate in the beginning on a control specification. We use state machine model to specify the behaviour of the control system. To start the specification we have to define the control relevant properties of inputs and outputs. We will use these properties to specify the behaviour. Let's "forget" for a while the details like numerical values of Pressure or allowed limits of the Actual Pressure value. Let's concentrate only on issues that determine the system behaviour.

First we analyze inputs and find names that describe well the required control properties:

- Command: we need a command to start the system; let's call it **Cmd_Start**.
- Required Pressure: we need the information that the parameter has changed; let's call it **RequiredPressure_Changed**.
- Actual Pressure: we need information whether the Pressure is ok or wrong; let's be specific defining 3 values: **Pressure_OK**, **Pressure_TooHigh**, **Pressure_TooLow**.
- Temperature Too High: we call this information **Motor_TooHot**.
- Timer: we need a signal when Timer elapses; we call it **Timer_OVER**.
- Counter: we need a signal when Counter reaches its limit; we call it **Counter_OVER**.

Then we define actions to be done:

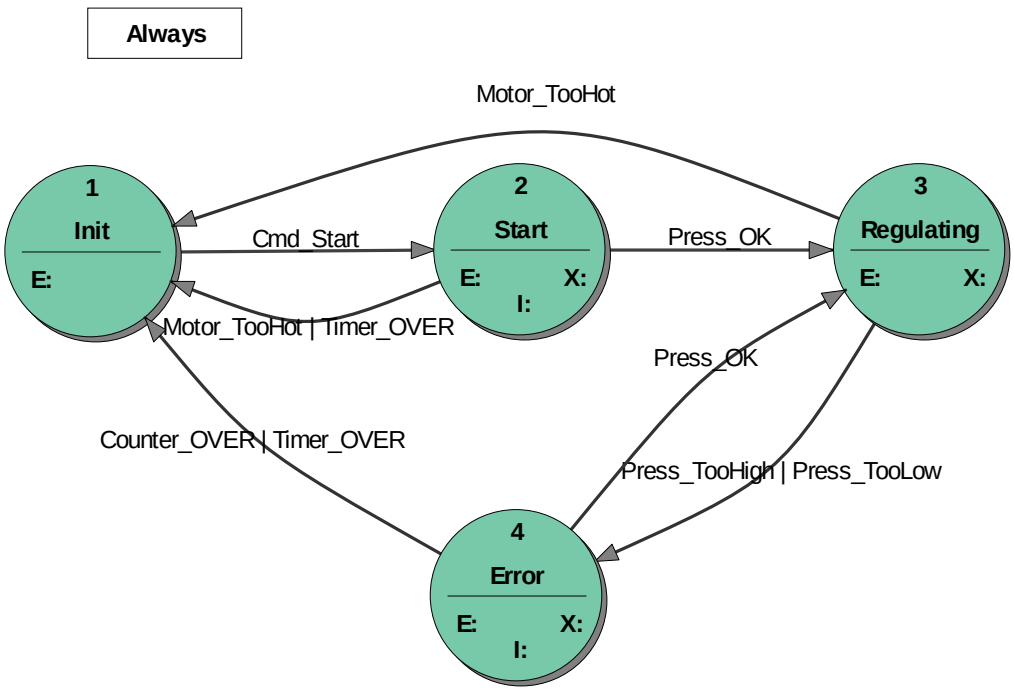
- Set Pressure: we need two signals which control this value; we call them **SetPressure_On** and **SetPressure_Off**.
- Pressure OK: we need two signals to control LED; we call them **LED_On** and **LED_Off**.
- Pressure Error: this is info for an operator; we call it **Pressure_Error**.
- Timer: we need a signal to start the Timer; we call it **Timer_ResetStart**.
- Counter: we need a signal to start the Counter; we call it **Counter_ResetStart**.

Using these definitions, we try to specify a state machine which realizes the requirements described in the first chapter. During the specification we may decide to add additional names. The first set of input and output names is just a starting point.

The basic component of a state machine is a state. After some consideration we could propose the following states:

- **Init**: where the state machine begins after start-up and returns after error.
- **Start**: the state machine sets the Pressure_Set, starts the Timer and Counter. The supervision starts, the state machines waits for the ActualPressure to reach the SetPressure value.
- **Regulating**: the ActualPressure is Ok and the state machine signals it by switching on the LED.
- **Error**: the ActualPressure is too low or too high and the state machine signals it by switching off the LED. If the ActualPressure value returns to the allowed range the state machine returns to the state Regulating, otherwise it goes to the state Init (controlled by the Timer and Counter).

Using these state names and previously defined input names we specify the following state transition diagram which should realize the required behaviour:



The state transition diagram shows the states, transitions and transition conditions. Thus, it gives a general idea about functioning of the state machine. To fully understand the state machine we have to see the actions of the state transition diagram. Normally, it is impossible to show them on a state transition diagram as it would be overloaded. The actions are seen in the state transition table. As an example we show here the state transition table of the state **Start**.

Start		E:	SetPress_Set Swip_On Counter_ResetStart Timer_ResetStart Ofun_CalcLimit
		X:	Timer_Stop
	RequiredPress_CHANGED		Timer_ResetStart
Regulating	Press_OK		
Init	Motor_TooHot | Timer_OVER		

In addition to the state transitions that have been displayed on the state transition diagram we see here the **E**ntry, E**X**it and Input actions. By entering the state the state machine sets the SetPressure value (SetPressure_Set) and starts the Timer and the Counter. We see also two additional Entry actions: Swip_On (switching on the supervision of the ActualPressure value) and Ofun_CalcLimit (calculation of limits for the supervision) which were introduced in course of detailed analysis of the requirements. There is also an Input action: when the RequiredPressure parameter changes the Timer is restarted.

The point is that we specified the behaviour of a control system using names representing control relevant input and output properties and states. This behaviour specification is a very stable component of the system: it is independent from the implementation or signal features (range, data type, scaling, offset). To be more specific, we do not care for instance how RequiredPressure_CHANGED or Pressure_OK or Ofun_CalcLimit come into being. We want to describe the behaviour using only information which influences it and we use for it expressive names. The implementation, representation, source of that information may change but it does not influence the system behaviour.

Real signals (objects)

The state machine specification described in the previous chapter is a “virtual” one. It uses abstract ideas in a form of names to describe behaviour. Eventually, we have to “link” the “virtual” names with real objects.

Any signal (we call it object because they are represented by RTDB objects) has two faces: the first one represented by control properties and the second one represented by object properties. The control properties are represented by names that describe features that may be used for control. We consider the control properties as “virtual” because they cannot be “seen”; they are just descriptions which make sense in the world of automata. In contrary to the control properties the object properties are “real”, physical features expressing time, voltage, units, etc.

In StateWORKS we do the link between the control and real properties using a Real Time Data Base (RTDB) which automatically filters out the control properties from real objects supplying the control information required by the “virtual” specification. This information can be than used by a standard Executor that realizes the state machine behaviour.

To describe the interface between the "virtual" environment of a state machine and the real objects let's take some examples:

- Motor_Temperature is a digital input which may be true or false (represented for instance by high and low voltage values). It is a true Boolean value. In the virtual environment the digital input is represented by two names: HIGH and LOW. Note that the two names represent an extension of a Boolean signal as this definition allows expression of four control properties:
    - HIGH – corresponds to the Boolean true,
    - LOW – corresponds to the Boolean false,
    - none – means "not known",
    - both HIGH and LOW present – not allowed.

  As the last control property does not make sense we have effectively three control properties.
- Timer is an object with the following "real" properties: Name, Const[1] (Timeout value) and Clock[1] (Time base). In the "virtual" state machine specification the Timer is represented by input control properties like: RESET, STOP, RUN, OVER and OVERSTOP and output control properties like: Reset, Stop, Start, ResetStart. These two "real" and "virtual" environments are completely different worlds. The real one with its Timeout value and Time base is useless and irrelevant for control purposes (the state machine behaves exactly in the same manner with a 1 sec timeout and a 1 hour timeout). The control properties in the virtual environment are very homogenous – they are just names describing control features. Those names are completely independent from the real features.
- Parameter is an object with the following "real" properties: Name, Category, Format, Unit, Low limit value, High limit value and Init value. In the "virtual" state machine specification the parameter object is represented by input control properties like: UNDEF, DEF, CHANGED and INIT (a parameter has no output control properties). Also in that case there is nothing common between the control properties required by the "virtual" state machine specification (names) and the "real" numbers, units, categories, etc.

StateWORKS Studio specification has several possibilities based on used state machine model and implemented RTDB objects. We cannot discuss them fully in this paper but we would like mention two specific StateWORKS features.

In general, a control system has a sequential part realized by the state machine and a combinatorial part that does not require the state machine functionality. In other words, the combinatorial part would have to be repeated in each state. StateWORKS offers a special solution for this problem. Instead of repeating combinatorial actions in all states we can write them in an Always section. In the Reg example we specify there the actions: Counter_ResetStart, Ofun_CalcLimit, SetPressure_Set if the Required Pressure value changes.

If we specify a state machine and the implementation system using StateWORKS Studio we use RTDB objects to define names of control properties. At this time the RTDB has 19 object types. If we are confronted with a problem that does not correspond to any of RTDB objects we can define our own object using the OFUN object. The OFUN object represents a software interface used to add missing functionality to RTDB. In the discussed Reg example the state

[1] These are the names used in ´StateWORKS Studio.

machine has to calculate the range of the ActualPressure value considered as correct. The range defined by two limits depends on the RequiredPressure value and must be recalculated any time the RequiredPressure changes. This task is realized by a C-function that is linked with the run-time system. In defining an OFUN object we specify the name of the function and its "owner". This information "binds" the RTDB with the C-function: the RTDB can call the function and the function can access the objects of its owner.

If you study the details of this example you will see the properties of all objects. Most of them are obvious and result from the requirements. Note the values for the SetPressure (NO) object:

- Scale Factor = 2047 / 1000 = 2.047
- Offset = 2048

and for the ActualPressure (NI) object:

- Scale Factor = 1000 / 4095 = 0.2442

Defining the RTDB objects we create any number of objects of a given type according to the state machine specification. Any specified state machine represents a specific type. We may create any number of the objects as different incarnations of the same state machine. They differ by their owned objects.

Run-time system

We can specify the state machine and the run-time system components using StateWORKS Studio. We can test the specified system using SWLab[2], SWMon and SWTerm. How far are we in this moment away from the implementation? Taking into consideration that SWLab is a real StateWORKS run-time system, SWLab is the implementation if we can use tcp/ip for signal transmission. It may be true for Command, Required Pressure, Pressure Ok and Pressure Error signals which are linked with the operator panel. It may be different for Set Pressure, Temperature Too High and Actual Pressure signals which are linked with the control peripheral devices. For them we have to write an IO-Unit which is an interface between the peripheral drivers and the RTDB. The IO-Unit has to be written in C++ and StateWORKS documentation provides a good description of class methods which allow RTDB access.

Conclusions

This paper tries to explain the way StateWORKS is used for implementation of control system, specifically for industrial control. The principle is to separate the behaviour from the implementation details. The behaviour is specified using state machine model and the result is an implementation independent specification. The StateWORKS concept assumes that the ultimate implementation will not be programmed. Therefore, the state machine specification must be complete – there will be no chance to add missing control later by coding. The specification in a form of state transition diagram, state transition tables or XML document is a perfect base for discussion, exchange and documentation.

The set of object types offered by the RTDB covers typical requirements for input and output signals. In some cases, where for instance some calculations are required, we shall use the OFUN object which represents the software interface to the RTDB.

[2] The NI and NO objects displayed in SWLab show the inputs (NI) of AD-converters and outputs (NO) of DA-converters in two forms: as a number and as an analog indicator. The analog indicators are labelled -10…+10 which is of course only one of possible interpretation of analog signals.

When you install the StateWORKS Studio you will find the entire project in the folder ..\Project\Examples-Web\Regulator. It may be tested using StateWORKS Studio.