Module 5 :: Delay and Loop blocks

The previous Module started with the statement "A programming language is not a programming language if it does not provide at least one statement which enables the use of an if-then-else structure." The same statement is valid for loop structures: A programming language is not a programming language if it does not provide at least one statement which enables the use of a loop structure.

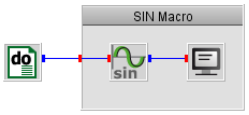
If you have studied the Tutorial from the beginning, you may intervene: We have used DO blocks and CLOCK blocks so often, aren't we through with loops in INSEL? No, we aren't. The blocks DO and CLOCK are T-blocks, not L-blocks. So, let us have a closer look at the difference between these two block groups.

No loop at all

Coming back to one of the most trivial examples of this Tutorial, where we have just calculated the sine of 45° on page 20. This example didn't use a timer at all. The CONST block, the SIN block, and the SCREEN were only called once. We could put the complete model into a macro with no inputs and no outputs.

One timer

If we wish that this macro (or model) depends on a variable input angle, we could add a DO block, delete the CONST block and connect the DO block's output with the sine block. This results in a loop.



Two timers

We could put this complete model into one macro gain, add an input to the DO block, and connect it to another DO block outside the macro, ending up in a nested DO block structure. There is no limit in nesting DO blocks and there is no limit in macro depth.

Three timers...

So we could continue in the same way, as long as we wish: Put the complete model into a macro again, add an input to the DO block, and so on.

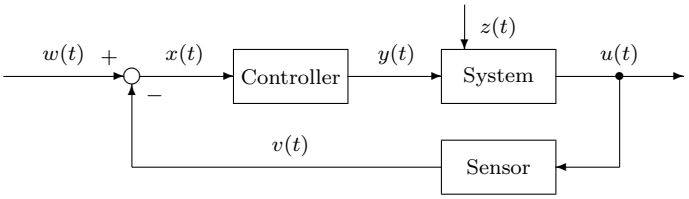
What we can learn from this simple example is, that Timer blocks can be used to create nested loop structures. But these loops always run over complete models, i. e. over all blocks which are connected to the respective timer's output and those blocks' successors. With this concept it is impossible to create local loops.

In other words, so far in this Tutorial we have treated "linear" simulation models only. Linear means here that any INSEL application we can write at this point follows basically a sequential structure, i. e. normally there is a Timer block which decides on the duration and time step of model execution and the rest of the model is executed more or less in a sequential order, except when there is an If block included, which allows to skip execution of some blocks depending of the conditions of the If blocks.

To express this fact in the language of structured programming, we have understood how we can handle sequential structures and if-then-else structures. The third required concept in structured programming is the concept of loops, which exactly is the topic of this Module.

5.1 Handling control cycles

Let us look at a control cycle which is typical in measurement and control technology.



The task of a control cycle is to keep a controlled process variable u within in a narrow range close to a given set point w. The variable u usually depends on w and a disturbance variable z.

At this point, we are not really interested in control strategies. Instead, we want to analyze the control cycle from a structural point of view. So let us assume that the values of the command variable w and the disturbance variable z are known. How can we perform a calculation of the cycle states?

The sum $x = w - v$ cannot be calculated because the sum depends on the output of the control process, i. e. the value of the feedback variable v, which is not yet known. Since the sum is unknown, the controller cannot be executed and therefore the values of u and v cannot be calculated. But the value of v is necessary to know when we want to calculate the sum x. So, what?

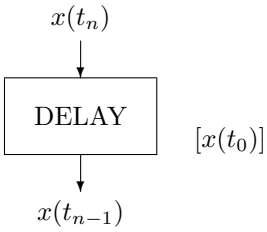
Algebraic loops

Closed loops like the one just described are called algebraic loops in computing. The solution of this problem is well known since the early times of analogue computing, i. e. when block diagram programming had its roots: Insert a delay element into the algebraic loop. So what is a delay element?

The characteristic properties of a delay element are that it delays its input signal for a specific time and, very important, that it is initialized with a value. This idea is the basis for a huge set of applications, ranging from numeric integration methods, numeric solutions of differential equations, and of course control cycles.

5.1.1 The DELAY block

In INSEL one delay element is a block from the group of Delay blocks named DELAY.



The DELAY block delays its input by one step. The optional parameter $x(t_0)$ is used as initial value. If not declared, $x(t_0)$ defaults to zero. The DELAY block can be found in the *Mathematics – Loops* category.

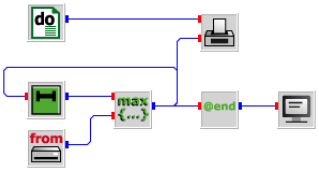
Controllers are typical Delay blocks in INSEL. And in fact, assuming that the controller starts with an initial value, let's say y_0, this simple measure solves our algebraic loop problem. Now that both inputs $y = y_0$ and z are known the system can deliver u and the sensor the required value v.

Exercise 5.1 In Module 11.4 you used the I-block MAXX (Absolute maximum) from the *Statistics – Maximum* category to find the overall maximum value for the hourly global irradiance on a horizontal surface in W/m^2 as saved in file `meteo82.dat`.

In the category *Mathematics – Basics* you can find an S-block named MAX (Maximum) which outputs the maximum value of its connected inputs.

Can you use this block to find the overall maximum radiation value, too?

Solution The solution makes use of a DELAY block, of course.

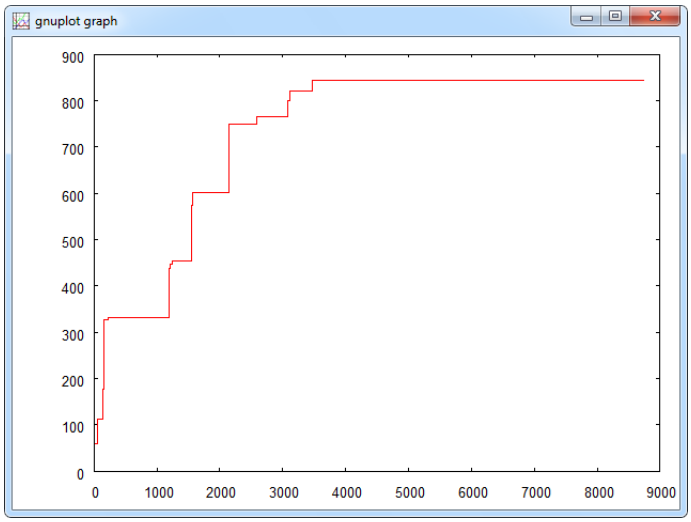


Since the time series starts at midnight, the radiation data are zero during the first calls and with an initial value zero of the DELAY block nothing happens. But when the first radiation value greater than zero occurs in `meteo82.dat`, the MAX block returns this value to the DELAY block which in return sets its output to this value.

In the next step the MAX block compares this output of the DELAY block with the next radiation value from the file. If the value from the READ block is greater than the output of the DELAY block, the MAX block returns this value to the DELAY block, otherwise the DELAY block receives its old value and nothing happens.

At the end of the simulation run the absolute maximum of the radiation time series is available at the MAX block's output.

In order to avoid too much SCREEN output, all data except the at-end value are filtered through the ATEND block. The next graph shows the evolution of the maximum with time.



It is interesting to have a look at the calculation list:

```
Number     Block        Group     Jump
-----------------------------------
     2     DO             T          1
     6     READ           S          1
     7     MAX            S          1
     1     ATEND          I          2
     3     SCREEN         S          1
     4     PLOT           S          1
     5     DELAY          D         -6
-----------------------------------
```

Please observe four details: (i) How the ATEND block jumps over its successor to the PLOT block, (ii) that the DELAY block points all the way up to the DO block as its successor, (iii) that we have used blocks from four different block groups in this simple exercise, and (iv) that the DELAY block is the last block in the calculation list, which is a typical property of all D-blocks. ■

Especially the last remark is worth a closer look. All blocks in INSEL depend on their inputs. This was one of the very first things we have learnt in Module 2. Now we learn, that Delay blocks are an exception to this rule. Why?

Constructor call

Delay blocks have an initial value at their output, before these blocks are called

for the first time. Is this a miracle? Of course not. INSEL has a mechanism called constructor call – similar to the destructor call we became acquainted with in Module 5, page 81. Before an INSEL model is executed by the inselEngine all blocks are called in this constructor call mode.

The constructor call is the time to check the plausibility of parameters fixed in the INSEL model. If for instance a value zero is provided as parameter of an attenuator block INSEL generates an error message and does not execute the model in order to avoid a division-by-zero exception. And this is the time to initialize the outputs of Delay blocks. But what happens when a model is executed?

Inputs as function of own outputs

The inselEngine must ensure, that all blocks which make direct use of the initial value of the DELAY block have a chance to access this value, and not the value after the DELAY block has been executed. So, in many cases all D-blocks appear at the end of the calculation list. In principle, it is possible to add a D-block to the calculation list, as soon as all blocks which make direct use of its output are already in the calculation list.

In the last example we have seen, that the output of the DELAY block is connected to a MAX block, which calculates the maximum on the basis of the DELAY block's output. The output of the MAX block is connected to the DELAY block as input. In consequence, this means that in case of the DELAY block its input depends on its actual output. This will become even clearer when we have a closer look at the group of L-Blocks later in this Module.

5.1.2 PID controller

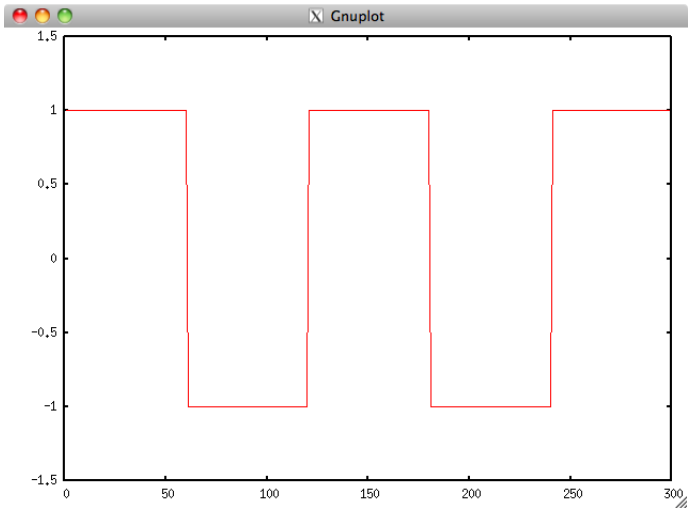
Coming back to control cycles, let us use a PID controller to follow a given signal, a step function, for instance.

What is a PID controller?

In order to prepare the solution, let us construct a demonstration signal.

Exercise 5.2 Construct an INSEL model for a step function which runs over five minutes in time steps of one second. The output signal shall vary between the values minus one and plus one with a sharp ramp, changing every 60 seconds.

Solution We have solved this problem by using two DO blocks, one ... see tutorial–ramp.vseit



idea 1: time axis

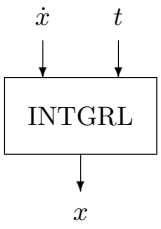
idea 2: plus one, minus one cahne via expg block

ides 3: introction of PID block

5.2 Solving differential equations

Sophisticated integration of differential equations. Long history before digital computing could overtake analogue simulation equipment

Maybe history, why digital block diagram simulation in the 60's practically had no chance against analogue computing - compared to today: extremely slow processors



Approx., x_0

$$\dot{x} = \cos(t) \Rightarrow x = \sin(t)$$

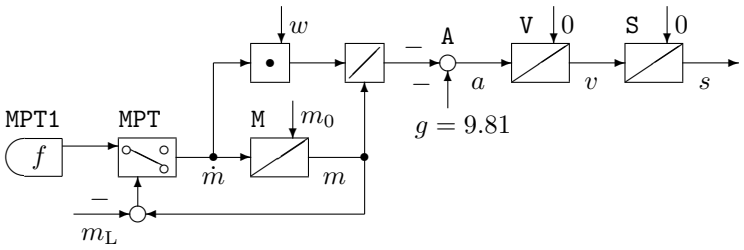
5.2.1 The Jentsch rocket

In his wonderful book "Digital simulation of continuos systems," published 1969, Jentsch [?] used the simple differential equation of a starting rocket to illustrate the principle of solving differential equations by the use of simulation languages. The equation is

$$a = -\frac{\dot{m}w}{m} - g$$

where a is the acceleration of the rocket, m the mass of the rocket (including gas), $\dot{m}$ the change in mass due to gas ejection, w is the velocity of the ejected gas relative to the rocket, and g the gravity of Earth, $g \approx 9.81\ \mathrm{m\,s^{-2}}$.

Since probably the younger readers of this Tutorial have never seen an "old-fashioned" block diagram description of a differential equation, here comes an adaption of Jentsch's example:



Starting from $\dot{m}$, the integrator `M` – a delay block – with initial value m_0 approximates the rocket mass m. The change in mass $\dot{m}$ is multiplied by w and divided by m by the two blocks marked with a dot and a division symbol. Finally, the acceleration a results from the summation block `A` – denoted by a small circle. By convention, the required minus signs are written close to the arrows pointing into the summation blocks. Velocity v and distance s are calculated by two more integrators named `V` and `S` with initial values zero.

Jentsch lets the example run through two blocks named `MPT1` and `MPT` over a time interval of twenty seconds.

Block `MPT1` determines the behavior of the ...

Block `MPT` represents a relay switches off ...

```
* --- Structure
      S    = I(0,V)
      V    = I(0,A)
      A    = -(MPT * W) / M - 9.81
      M    = I(M0,MPT)
      MPT  = REL(M - ML,MTP1)
      MPT1 = KUL(KLMPT1)
```

```
* --- Parameters
      W      = 3000
      MO     = 3300
      ML     = 300
      KLMPT1 = 0,-160, 20,-160
* --- Processing
      TIME   = (0/0.1,20)
      PRTIME = (0/0.1,20)
      PRINT(1,1) M,V,S / 1000
         FORMAT 1 (2(1) / 3)
         HEAD 1   (M,KG,V,M/S,S,KM)
      PLOT(2,1) M,V,S / 1000
         FORMAT 2 (0,4000,3/0,1.E+4,4,4/0,60,5)
      END
         KLMPT1 = 0,-320, 18.75,0
      END
      STOP
```

Exercise 5.3 Can you convert Jentsch's rocket example into an INSEL model?

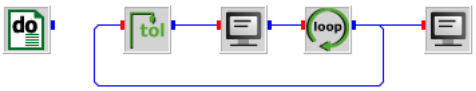
Solution ■

5.2.2 Solar collector equation

5.3 Loop block concept

Explain LOOP, NULL, and MPP.

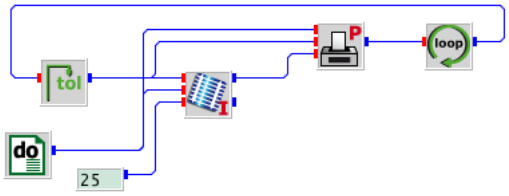
loop.vseit



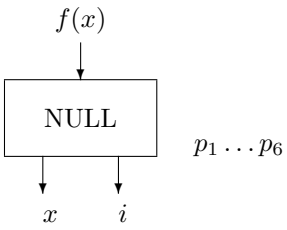
A LOOP block and a TOL block are connected in a loop. The LOOP block uses 1 as initial value, 3 as final value and 1 as increment. Two SCREEN blocks display the outputs of the TOL and LOOP block, respectively.

```
   1.0000000
   2.0000000
   3.0000000
Final output    3.0
```

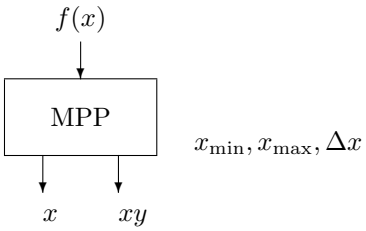
```
   1.0000000
   2.0000000
   3.0000000
Final output    3.0
```



Example: NULL block - root of a function, involution algorithm, regula falsi algorithm.



More applied: maximum power point calculation.



Mention only: Even more applied: Battery charge regulator – see Module 7.2

Loop Blocks and Iterations

Iteration blocks are called Loop block or short L-Blocks. In INSEL the iteration blocks are the LOOP, MPP and NULL block.

- The LOOP block runs through a sequence of values defined by parameters, restricted to a part of the simulation model.

- The NULL block searches a root of a continuous function.

- The MPP block simulates an ideal maximum power point tracker. In general, the MPP block can be used to find the maximum of any unimodal function.

The output of an L-block must always be the input of a TOL (top of Loop) block.