Simulation Practice
with
Simulink

SS 2015
03.03.2015

Prof. Dr. Alfred Irber
University of applied sciences Munich
Introduction to Matlab

1.1 Variables

1.2 Vectors and Matrices

1.3 Workspace

1.4 Grafical functions

1.5 Matlab-Programming

1.5.1 Matlab-Scripts

1.5.2 Matlab-Funktionen

2 Introduction to Simulink

2.1 Developing of a simulink block diagram

2.2 Simulink-Simulation

2.3 Solutions of DE's with Simulink

2.3.1 Demonstration of the solver Euler(ode1)

2.4 Subsystems

2.4.1 Virtual Blocks

3 Automatic Code Generation

3.1 Modeling the system

3.2 Simulation

3.3 Code Generation

3.4 Execution of the generated code on the microcontroller

3.5 Partitioning microcontroller - Simulink-Model
Introduction to Matlab

- Numeric Tool for Calculation und Simulation
- Name derived from MATrix LABoratory

1.1 Variables

>> x=3.5  (Without semicolon output to Command Window, Termination with semicolon suppresses the output !!!)

- No previous declaration necessary
- All variables have the type double by default

Data types:
int8, uint8, int16, uint16, int32, uint32, int64, uint64
single 32-Bit
double 64-Bit
logical true, false
char strings
cell arrays
structure

Example:
>> y= uint8(3)
y=3
>> name= 'Alfred'
name = Alfred

1.2 Vectors and Matrices

- In the strict sense all variables are matrices
- With the instruction "y=3" a 1x1-Matrix is created!!!!

- Separators within a row are spaces and commas
>> v = [1 5 -3] or [1,5,3] vector

The rows are separated by semicolons:
>> M = [1 3; 4 0; 2 3] 3x2 – matrix (3 rows, 2 columns)

- Therefore a pure column-vector is defined by:
>> Spv = [1; 2; 3] 1x3 -matrix

Large vectors with constant step size can be defined as follows:
e.g. vector for simulation time:
>>t=[0:0.1:1]
t = 0 0.1000 0.2000 0.3000 0.4000 0.5000
0.6000 0.7000 0.8000 0.9000 1.0000
Also possible, but not known as well:

\[
\begin{array}{cccccc}
  \text{r} &=& 0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

Matrices and Functions

Effect from component to component:

\[
\begin{array}{cccccc}
  \text{t} &=& 0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
  \text{s} &=& 0 & 0.8415 & 0.9093 & 0.1411 & -0.7568 & -0.958 \\
\end{array}
\]

Each element of s is calculated by applying the formula \( \sin(t) \).

Access on single elements

Access is realized by using the index in round brackets:

\[
\begin{array}{cccccc}
  \text{x} &=& \text{s}(3) \\
\end{array}
\]

\[
\begin{array}{ccccccc}
  \text{s} &=& 0 & 0.8415 & 0 & 0.1411 & -0.7568 & -0.9589 \\
\end{array}
\]

Access on single rows/columns of a matrix

Especially to output single rows or columns, the colon-operator must be used properly. In this case all elements of the corresponding row/column are addressed by a single colon:

\[
\begin{array}{ccccccc}
  \text{mat} &=& \begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \end{bmatrix} \\
\end{array}
\]

\[
\begin{array}{cccc}
  \text{ans} &=& \begin{bmatrix} 5 & 6 & 7 \end{bmatrix} \\
\end{array}
\]

Operations on matrices

Without any appendix all operations are matrix operations:

\[
\begin{array}{ccccccc}
  \text{M} &=& \begin{bmatrix} 1 & 2 & 3 \\ 4 & -1 & 2 \end{bmatrix} \\
  \text{N} &=& \begin{bmatrix} 1 & 2 & -1 \\ 4 & -1 & 1 \end{bmatrix} \\
\end{array}
\]

\[
\begin{array}{ccccccc}
  \text{V} &=& \text{M} \ast \text{N} \\
\end{array}
\]

\[
\begin{array}{ccccccc}
  \% \text{V} &=& \begin{bmatrix} 15 & 0 & 4 \\ 4 & 9 & -3 \end{bmatrix} \\
\end{array}
\]
**Field operations**  
In addition to matrix-operations arithmetic operations are necessary, which are done component by component:  

*Scalar product .*  

Number of rows and columns for M must also be equal to rows and columns in N !!  

```matlab
A=[1 2 3];
B=[4 5 6];
C= A .* B
%C =
%  4  10  18
```

### 1.3 Workspace

- `>> who` delivers the name of the stored variable  
- `>> whos` returns detailed information  
- `>> clear` clears the workspace

Example:

```matlab
>> clear
>> x=3
>> who
```

Your variables are:

```matlab
x
```

```matlab
>> whos
Name      Size           Bytes  Class       Attributes
          1x1             8  double
```

**Workspace Browser**  
A very useful overview is also delivered by the Workspace Browser

![Workspace Browser Image](image)
1.4 Graphic functions

>> clear
>> t=[0:.1:2*pi];
>> s=sin(t);
>> grid on;
>> plot(t,s)

There are a lot of additional graphic functions. The most commonly used are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axis(xmin, xmax, ymin, ymax)</td>
<td>sets the limits for the x- and y-axis</td>
</tr>
<tr>
<td>axis('auto')</td>
<td>automatic axis-scaling</td>
</tr>
<tr>
<td>grid [on</td>
<td>off]</td>
</tr>
<tr>
<td>zoom[on</td>
<td>off]</td>
</tr>
<tr>
<td>xlabel(string)</td>
<td>labeling the x-axis</td>
</tr>
<tr>
<td>ylabel(string)</td>
<td>labeling the y-axis</td>
</tr>
<tr>
<td>title(string)</td>
<td>creating a title</td>
</tr>
<tr>
<td>text(x,y,string)</td>
<td>locating a string</td>
</tr>
<tr>
<td>legend(string1,string2…)</td>
<td>creating a legend</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Colour Specifiers</th>
<th>Marker Specifiers</th>
<th>Line Style Specifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>k black</td>
<td>. points</td>
<td>- solid line (default)</td>
</tr>
<tr>
<td>b blue</td>
<td>o circle</td>
<td>-- dashed line</td>
</tr>
<tr>
<td>r red</td>
<td>* asteriks</td>
<td>-. dash-dot line</td>
</tr>
<tr>
<td>g green</td>
<td>+,x plus, cross</td>
<td>: dotted</td>
</tr>
</tbody>
</table>

Plot-Demo:

>> t=[0:0.01:2];
>> sinfkt= sin(2*pi*5*t);
>> cosfkt=2*cos(2*pi*3*t);
>> expfkt= exp(-2*t);

Plot-commands

>> clf %clear
>> hold on %hold on old plot
>> grid on
>> title('Plot-Demo')
>> xlabel('Time t[s]')
In addition to a numeric tool, Matlab is a separate programming language too. Its structure is similar to the C programming language. For details, look at other literature, or the Online-Help of Matlab.

1.5 Matlab-Programming

Besides putting in instructions at the command window, sequences of Matlab commands can be stored in so-called Matlab-scripts. These are simple text-files with the extension ‘.m’, therefore they are called m-files. This simple form of an m-file is also referred to as a procedure.

The previous plot-example as a Matlab-procedure:

```
%Plot-Demo
clf
hold on
grid on
title('Plot-Demo')
xlabel('Zeit t[s]')
legend('sin','cos','exp')
plot(t,sinfkt,'k-', t,cosfkt,'b--',t, expfkt,'m.');
```

- Matlab-Procedures are also often just used to define a variable in the Workspace.

```
z.B script1.m
clear
A=3;
```
B=[0:0.1:10];

### 1.5.2 Matlab-Functions

Much more flexible than the pure summarization of Matlab-commands in script-files is to define Matlab functions, so you can pass parameters to these. The name of the function and its corresponding file must be the same, because Matlab is looking for an m-file with the identical name. A function is defined by the keyword `function` in the first line of the file. The plot-example as Matlab-function:

```matlab
% Plot-DEMO as function
function [t, sinfkt, cosfk, expfkt] = funkbsp_1(f1, f2, damp)
% funktion funkbsp1
% Call: [t, sinfkt, cosfk, expfkt] = funkbsp_1(f1, f2, damp) % or funkbsp_1(f1, f2, damp)
% description of the in-and output -parameters

t=[0:0.01:2];
sinfkt= sin(2*pi*f1*t);
cosfkt=2*cos(2*pi*f2*t);
expfkt= exp(-damp*t);

% Plot
clf
hold on
grid on
title('Plot-Demo')
xlabel('Zeit t[s]')
legend('sin', 'cos', 'exp')
plot(t,sinfkt, 'k-', t, cosfkt, 'b--', t, expfkt, 'm.');
```

Function call from workspace:

```matlab
>> clear
>> f1=2; f2=3; damp=5;
>> [t, sinfkt, cosfk, expfkt] = funkbsp_1(f1, f2, damp);
>> whos
Name           Size            Bytes  Class          Attributes
----------    ---------            ------  --------    ------------
c1            1x201              1608  double          
e1            1x201              1608  double          
s2            1x201              1608  double          
zeit          1x201              1608  double          

Of course, instead of passing constant values, also variables can be passed to the function. If there is no interest in the calculated vector, there is no need to have a return-vector.
```
2 Introduction to Simulink

Simulink is graphical user interface for modeling and simulation of dynamic systems. The graphical user interface allows the representation of the considered system as a block diagram. Simulink also includes a comprehensive block library for linear, non-linear and discrete systems. These systems can be described with differential equations as well as difference equations in the case of discrete systems. The numerical solution of these equation systems is an essential part of Simulink.

2.1 Developing of a Simulink block diagram

Using the library blocks following block diagram can be build easily.

2.2 Simulink-Simulation

The simulation parameters are set up by using the menu command Simulation – Configuration Parameters. Thereby a very important step is the pane Solver. A solver is the special mathematical procedure to solve the differential equation in a numerical way.

At the Solver options/Type we choose Fixed-step instead of Variable-step. So the desired constant sample time e.g. (0.1 sec) can be set.

As solver ode1(Euler) should be chosen, which will be explained precisely later.

All remaining parameters should be accepted unchanged and will not be further discussed here.
After setting the simulation parameters, the simulation can be started by pushing the play-button, which results in the following plausible result:

We also can store the result in a Matlab-variable and watch the plot with Matlab. For the scope the following settings are necessary.
In Matlab the workspace can be reviewed by using the `whos` command as well as the Workspace-Editor

```matlab
>> whos
Name               Size           Bytes  Class     Attributes
S_test1_Signale    101x3          2424  double
tout               101x1          808   double
```

With the following call of `plot`, the graphics are displayed by Matlab:

```matlab
>> plot(S_test1_Signale(:,1), [S_test1_Signale(:,2),
                              S_test1_Signale(:,3)])
```

Note: A single colon addresses all elements of a corresponding column.

### 2.3 Solutions of DE's with Simulink

To solve Differential Equations (DE’s) using Matlab is possible, but relatively laborious. The solution with Simulink is much simpler. The trick is to represent the DE in form of a block diagram.

A simple example is the voltage curve of the capacitor in case of an RC-Circuit:
Ue = Ur + Uc
I = C* dUc/dt
Ue = RC* dUc/dt + Uc
dUc/dt + 1/RC * Uc = Ue/RC
For this DE, the following solution is well known:

Uc = Ue * (1-exp(-t/(RC))

The idea is to solve the equation with Simulink as follows:
By integration of the deviation-function Uc'(t) of the required solution using the Simulink integration block and multiplication with the factor 1/RC, we get the function of the solution, which can be subtracted.
So we get the following diagram:
The scope shows the step function $U_e(t)$, which jumps at time $t=0$ from 0 to 1, the current $I$, which has got the value $U_e/R=0.1$ at time $t=0$ and the voltage $U_c$ at the capacitor, which starts at 0 and approaches asymptotically the value $U_e=1\, \text{V}$.

For comparison, beside it the solution with Simulink using the mathematical solution for $U_c= U_e/R(1-\exp(-t/Tau))$ with $\tau = R\cdot C = 47\times 10^{-6}\times 1\times 10^4 = 0.47\, \text{s}$ is plotted as well.

The comparison shows the excellent correlation of the mathematical and the numerical solution with Simulink.

### 2.3.1 Demonstration of the solver Euler(ode1)

The perhaps most simple procedure of the numerical solution of DE’s in the case of initial value problems is the polygonal line procedure by Euler. It is based on the idea, that the size of the integral from $t_0$ to $t_0+dt$, is represented by a rectangle, whose height is equal to the function value at the left side of the integration interval and whose integration step $dt$. So we get: Integral $I= \int_{t_0}^{t} f(t) \, dt$

![Diagram showing the Euler method](image)
This procedure can be implemented by software quite simply. For practice and because of the powerful plot function, the procedure is realized by Matlab as follows. Please be aware, that all indices in Matlab will start with 1!

Knowing \( U_c(1)=0 \) the value \( I(1)= (U_e-U_c(1))/R =U_e/R \) can be calculated.

For the time period \( dt \), the capacitor is charged by this approximately constant current to the voltage \( U_c=I(1)^*dt/C \). At the next step this voltage \( U_c \) is subtracted from \( U_e \), whereby charge current is already reduced. This reduced current is charging the capacitor at the next step and so on.

\[
\% \text{The polygonal-line procedure for the solution of linear DE's}
\%
\text{Demonstration of the switch-on behaviour (characteristic) of a RC-Circuit}
\]

\[
R=1E4;
C=47E-6;
U_e=1.0; %V \quad \text{Input voltage}
U_c=0;25; %V \quad \text{voltage at the capacitor}
\]

\[
t=0:0.1:2.5;
U_c(1)=0; %V \quad \text{starting voltage}
I=0.0; %A \quad \text{charge current}
h=0.1; %s \quad \text{step size}
\]

\[
\% \text{The polygonal-line procedure}
\%
\text{The altitude of the rectangle is value at the left side}
\text{disp('Start')}
\text{for i=2:26}
\quad I = (U_e - U_c(i-1))/R; %Calculation of the current
\quad U_c(i) = U_c(i-1) + I*h/C; %Integration of the current(summation)
\text{end}
\%
\text{Output}
\text{title('switching-on procedure at the capacitor of an RC-circuit')}
\text{legend('Euler,step size h=0.1','Math exact', 'current line I/Io',3)}
\text{xlabel('t[s]')}
\text{ylabel('U_c(t)/U_e')}
\text{axis([0,2.5,0,1.0])};
\text{plot(t,U_c,'k:');}
\text{hold on} % don’t remove current plot
\text{Comparison with numerical solution}
\text{plot(t,1-exp(-t/(R*C)),'k-'); %voltage}
\text{plot(t,exp(-t/(R*C)),'k--'); %current}
\%
\text{End}
You have only to reduce the step size $h$ from 0.1 to 0.01 to get optically indistinguishable curves between the numerical and the mathematically exact solution.

**Further solvers:**
The further integration procedures are all based on an improvement of precision of the integration, by approximating the integral no longer by a rectangle but by more exact procedures, which are taking into account further deviations of the curves. Look at Runge-Kutta-procedure and so on. Additionally not only initial value problems but boundary value problems can be solved by Matlab.
2.4 Subsystems

To represent more clearly arranged models, Simulink offers the establishment of a hierarchical block structure by grouping blocks into subsystems.

You can create a subsystem in two ways:

1. All function blocks, which should be converted to a subsystem are to be selected individually or by using a bounding box. Choosing Create Subsystem from the Edit Menu replaces the selected blocks with a subsystem block. The following diagram shows the result.

2. From the submenu Ports & Subsystems library you copy a subsystem-block in the model and insert the functionality you want in the subsystem.
2.4.1 Virtual Blocks

In Simulink you can distinguish between two kinds of blocks:
Virtual blocks and Non-virtual blocks

Non-Virtual blocks: Non-virtual blocks play an active role in the simulation of a system.

Virtual blocks: They help to organize a model graphically.

Some Simulink blocks are virtual in some circumstances and non-virtual in others. Such blocks are called conditionally virtual blocks.

Blocks such as multiplier, demultiplexer, scopes and displays are always virtual.
3 Automatic Code Generation

In general modeling and simulation are pre-stages to realizing a system. In most cases C-code is only to be generated from some parts of a system, whereas the rest of the system is modeled only to test this part of the system. For the automatic code generation the special Simulink blockset **Real-Time-Workshop (RTW)** is necessary.

With automatic code generation, C-code is created, which is running usually on a microcontroller. The part of the system to generate C-code has to be located in a subsystem.

3.1 Modeling the system

Example: Logical AND-block

The diagram sim_AND_Glied_2 shows the complete system, which is composed of the subsystem and additional components necessary for the simulation. The subsystem contains the AND-block, which should be realized by C-code. The 2 Constant-blocks provides the input signals, the display block shows the result of the simulation.

3.2 Simulation

For testing, the simulation can be executed for the input signals (1,1) and (0,0) as well as (1,0) and (0,0).
3.3 Code Generation

Configuration of the RTW

Before the actual code generation takes place, the RTW can be configured by diverse settings. This is done using the panel Tools/Real-Time Workshop/ Options...

With the panel Real-Time Workshop/ Target Selection/System target file, the code generation can be optimized for special requirements. Because at this moment we don’t have any special requirements, we select ert.tlc RealTime Workshop Embedded Coder (no auto configures).
As *Language* we select **C** of course. Furthermore the generation of an HTML-report is recommended, because with that the assignment of code and function blocks can be traced in a very simple way. Therefore in *Documentation* all 3 points are selected.

Because we only want to generate C-code, the option *Generate makefile* is to be disabled and the option *Generate Code only* is to be selected.

With the subpanels of *Real-Time Workshop* only settings have been modified for *Interface* and *Hardware Implementation*.

For *Interface* the subpanel *Software environment* is to be checked. Under *Support* only the numeric data types need to be selected. In the current case, no floating point numbers, no non-finite numbers (e.g., NaN, Inf) and no complex numbers. Furthermore attention should be paid, that the point *non-inlined S-functions* – as provided per default – is deactivated, except if non-inlined S-functions are actually used.

For *Hardware Implementation* the used processor type – in our case **Infineon C16x** – has to chosen. All other points can be taken over without any changes.
Start of Code Generation
For the actual code generation the subsystem has to be selected and in the pop-up menu the sub-item Real-Time Workshop/Build Subsystem ...has to be called. So the dialog Build Code for Subsystems, which shows a list of all subsystem parameters, will be opened. For our AND-example this list is empty. The code generation is now started by pushing the Build-Button. In this way a list of all workspace variables appears, which are needed for the system to be generated. Because the system is simulated by constants, this list is also empty for our example. After starting the code generation by pressing Build-Button, the generation process can be observed.

Directories and Files built by code generation
If the code generation has been successful, the sub-directories slprj und AND0_ert_ertw are produced in the current directory.

The directory slprj is a project-directory and should not be considered any longer.
The directory AND_ert_rtw (Model_ert_rtw) contains the following files:
To get the generated code running on a microcontroller e.g. Infineon C166, the selected files AND0.c, AND0.h, AND_private.h AND0_types.h und rtwtypes.h are needed. Ert_main.c is not needed, because the main-program of the micro-controller should to be used. The generated files have got the following content:

```c
/*
 * File: S_AND.h
 *
 * Real-Time Workshop code generated for Simulink model S_AND.
 *
 */
#include <stddef.h>
#include "rtwtypes.h"
#include "S_AND_types.h"

/* External inputs (root inport signals with auto storage) */
typedef struct {
    boolean_T In1;                      /* '<Root>/In1' */
    boolean_T In2;                      /* '<Root>/In2' */
} ExternalInputs_S_AND;

/* External outputs (root outports fed by signals with auto storage) */
typedef struct {
    boolean_T Out1;                      /* '<Root>/Out1' */
} ExternalOutputs_S_AND;

/* External inputs (root inport signals with auto storage) */
extern ExternalInputs_S_AND S_AND_U;

/* External outputs (root outports fed by signals with auto storage) */
extern ExternalOutputs_S_AND S_AND_Y;

/* Model entry point functions */
extern void S_AND_initialize(void);
extern void S_AND_output(void);
extern void S_AND_update(void);

/* Real-time Model object */
extern RT_MODEL_S_AND * S_AND_M;
```

The file S_AND.h contains type definitions for input- and output signals. All inputs are composed by the type definition ExternalInputs_S_AND. In our example these are the input signals In1 und In2 of the AND-subsystems.

typedef struct {
    boolean_T In1;
    boolean_T In2;
} ExternalInputs_S_AND;

All outputs are combined by the ExternalOutputs_S_AND typedef. In our case we just have the output-signal Out1.

typedef struct {
    boolean_T Out1;
} ExternalOutputs_S_AND;

Furthermore the prototypes of the function, which are defined in S_AND.c, are given:

extern void S_AND_initialize(void);
extern void S_AND_step(void);

As well the global variables are exported, which are defined in \texttt{S_AND.c}

```c
/*
 * File: S_AND.c
 *
 * Real-Time Workshop code generated for Simulink model S_AND.
 *
 */
#include "S_AND.h"
#include "S_AND_private.h"

/* External inputs (root import signals with auto storage) */
ExternalInputs_S_AND S_AND_U;

/* External outputs (root outports fed by signals with auto storage) */
ExternalOutputs_S_AND S_AND_Y;

/* Model step function */
void S_AND_step(void)
{
    S_AND_Y.Out1 = (S_AND_U.In1 && S_AND_U.In2);
}

/* Model initialize function */
void S_AND_initialize(void)
{
    /* initialize error status */
    rtmSetErrorStatus(S_AND_M, (const char_T *)0);

    /* external inputs */
    S_AND_U.In1 = FALSE;
    S_AND_U.In2 = FALSE;

    /* external outputs */
    S_AND_Y.Out1 = FALSE;
}
```

The generated code is located in \texttt{S_AND.c}.

First the global input variables \texttt{ExternalInputs\_S\_AND S\_AND\_U} and the output variables \texttt{ExternalOutputs\_S\_AND S\_AND\_Y} are defined.

In the function \texttt{S\_AND\_step}() the functionality of the model is realized, i.e. the logical conjunction of both input signals and the result is assigned to the output. Bear in mind that the function doesn’t have any parameters, because global variables are used.

Later on this function has to be called from the surrounding microcontroller-program periodically e.g. in 1ms in order to assure a real-time ability.

Futhermore \texttt{S\_AND.c} includes the function \texttt{S\_AND\_initialize()}, which can be called in the initialize phase and provides all variables with appropriate values.
To be able to be completely compiled, additionally the header-files S_AND_types.h, S_AND_private.h and rtwtypes.h are needed.

In rtwtypes.h all of the types are defined, which are necessary for a special target e.g. Infineon C166: For instance in S_AND.h the type boolean_T is used.
For the target processor by using the typedef unsigned char boolean_T boolean_T is returned to unsigned char, which is compatible with C.

The files S_AND_types.h und S_AND_private.h include the datatype RT_MODEL_S_AND, which is necessary for the error output of the macro rtmSetErrorStatus(..) in S_AND.c

### 3.4 Execution of the generated code on the microcontroller

The execution of the generated code will be performed on the microcontroller-board C166, which is known from the lecture ‘embedded systems’.
First the simulation tool µVision is an adequate platform.
As a starting point we can use a project, which is able to read and set buttons by using IO-ports.
This system is dealt with in the practical part v2 of the embedded systems lecture and should be the basis for our project.

1. Including autocode-files into the µVision-project
S_AND.c, S_AND.h, S_AND_private.h, S_AND_types.h and rtwtypes.h have to be copied in the source directory and added to the µVision-project

2. Modification of the microcontrollers MAIN.c
Here the essential contents of the file Main.c are presented, which was modified compared to the original µVision-Project. Following steps have to be done:

- Including the header files
- Call of the function S_AND_initialize() before the while(1)-loop to initialize the input- and output-signals once only.
- In the continuously running while-loop the state of the buttons is saved in the variables S1 and S2. At each press of the buttons the value of the variables S1 and S2 is to be inverted.
- By the call of InputData() the state of the buttons is transferred to the input variables In1/ In2 of the model.
- By the call of S_AND_Step() the application part, in this case the conjunction of the logical variables is performed.
- At the end of the while-loop the result of the function S_AND_step() is returned to the microcontroller-hardware.
//File MAIN.C of the microcontroller-project
//Autocode-Includes
#include <math.h>
#include <stddef.h>
#include <limits.h>
#include "rtwtypes.h"
#include "S_AND.h"
#include "S_AND_private.h"
static bit S1=0;
static bit S2=0;
void main(void)
{
    
    //Initialization of the Auto-Code
    S_AND_initialize();

    while(1)
    {
        if(KeyDown()) {
            //Reading in the buttons
            if(!(uiTasteglob & 0x10)) //Button4
            {
                S1 =~ S1;
            }
            if(!(uiTasteglob & 0x20)) //Button5
            {
                S2 =~ S2;
            }

        //Auto-Code
        //Reading in of the variables
        //State of the button --> Model
        InputData();

        //Call of the AutoCode-functionality
        //processing of the buttons read in
        S_AND_step();

        //output to the microcontroller-ports
        OutputData();
    }
}

void InputData(void)
{
    S_AND_U.In1= S1;
    S_AND_U.In2= S2;
}

void OutputData(void)
{
    P1L_4=S_AND_Y.Out1;
}
3.5 **Partitioning microcontroller - Simulink-Model**

The **microcontroller system** has to provide all of the peripheral devices and the appropriate drivers.

The **Simulink-Model** can take over the logical processing of the data. At this point the advantages of the graphical programming with **Simulink** can be used to the full.