1. Develop MATLAB Model, Code generation, MIL, SIL test. Document all results.

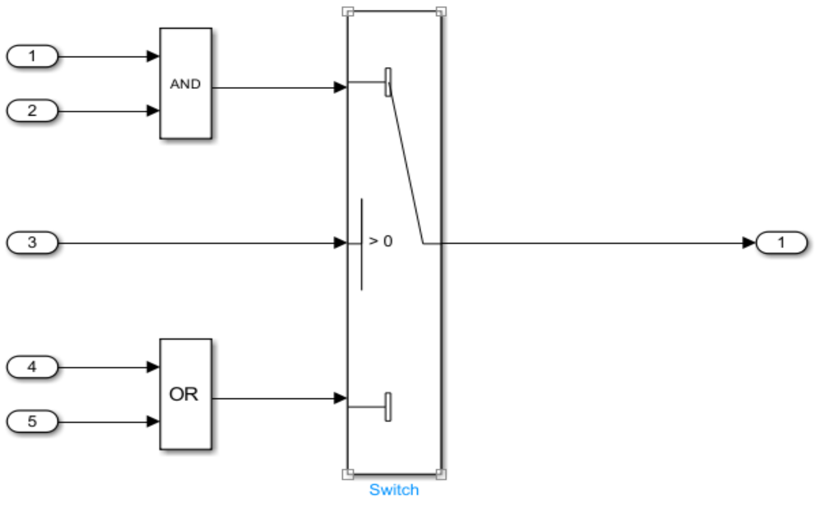
❖ Requirement :

```
if (x = 1)
        o/p = a (AND) b;
else
        o/p = c (OR) d;
```

❖ Pseudo code :

```
#include <stdio.h>
int main()
{
int a,b,c,d,r,s;
if (s == 1)
   r = a & b;
else
  r = c | d;
}
```

❖ Model :



❖ Code :

```
#include "task1.h"
/* External inputs (root inport signals with default storage) */
ExtU_task1_T task1_U;
/* External outputs (root outports fed by signals with default storage) */
ExtY_task1_T task1_Y;
/* Real-time model */
static RT_MODEL_task1_T task1_M_;
```

```
RT_MODEL_task1_T *const task1_M = &task1_M_;
/* Model step function */
void task1_step(void)
{
/* Switch: '<S1>/Switch' incorporates:
*  Inport: '<Root>/Inport4'
*/
if (task1_U.Inport4 > 0.0) {
/* Outport: '<Root>/Outport' incorporates:
*  Inport: '<Root>/Inport'
*  Inport: '<Root>/Inport1'
*  Logic: '<S1>/Logical Operator'
*/
task1_Y.Outport = (task1_U.Inport && task1_U.Inport1);
}
else {
/* Outport: '<Root>/Outport' incorporates:
*  Inport: '<Root>/Inport2'
*  Inport: '<Root>/Inport3'
*  Logic: '<S1>/Logical Operator1'
*/
task1_Y.Outport = (task1_U.Inport2 || task1_U.Inport3);
}
/* End of Switch: '<S1>/Switch' */
}
/* Model initialize function */
void task1_initialize(void)
{
/* (no initialization code required) */
}
/* Model terminate function */
void task1_terminate(void)
{
/* (no terminate code required) */
}
/*
* File trailer for generated code.
*
* [EOF]
*/
```

❖ <u>MIL Testing :</u>

MIL (Model-in-the-Loop) testing in MATLAB refers to the process of verifying and validating control algorithms or system models within a simulated environment before implementing them in hardware.

- To verify the logic, performance, and expected behavior of a system before moving to actual hardware.
- To identify and fix errors early in the design process.

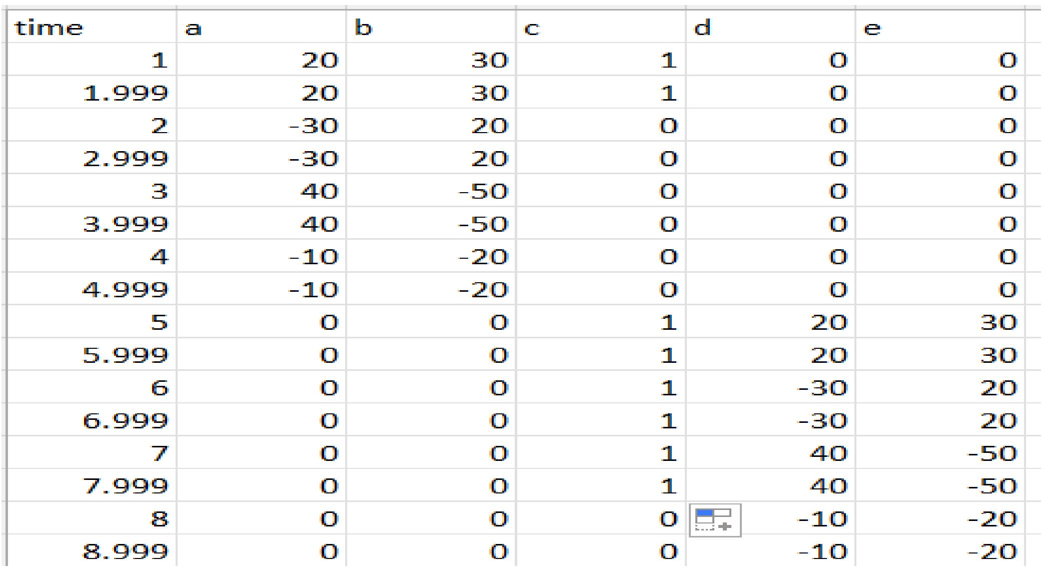
❖ Procedure for MIL :

To achieve we follow the below steps :

1. Initially Make sure that we know the functionality in the requirement system.
2. In this model we are dealing with switch, and, or blocks.
3. As we have 5 inputs as per the requirement (i.e control input, and , or -> 2 i/p)
4. Therefore in signal builder we require 5 inputs.

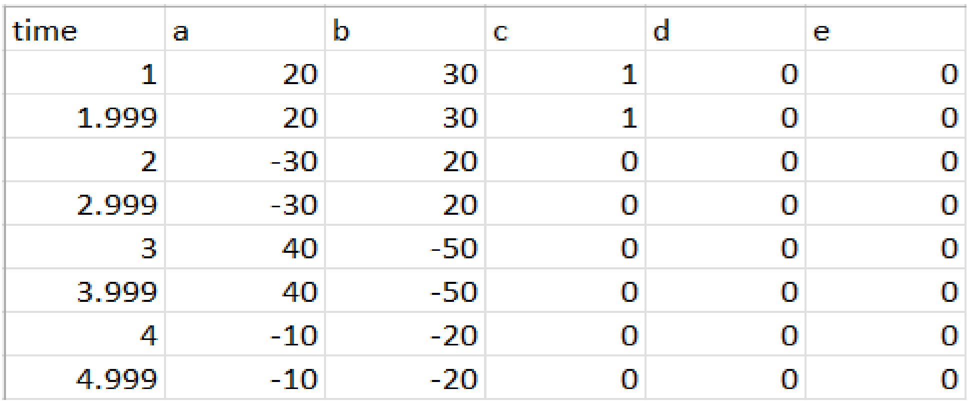
   Signal builder : The **Signal Builder** block in Simulink is used to create, edit, and manage **custom input signals** for testing models. It allows users to design various **waveforms, step changes, pulses, and other signal patterns** without needing external files or MATLAB scripts.

5. Double click on Click on signal builder , select -> import from file -> Browse for the Path and choose the appropriate(other than c drive is preferred) -> placement of data as Replace existing datasheet -> confirm selection -> apply -> ok.
   Here the imported file is excel sheet where u give custom inputs to check the functionality in MIL)



time	a	b	c	d	e
1	20	30	1	0	0
1.999	20	30	1	0	0
2	-30	20	0	0	0
2.999	-30	20	0	0	0
3	40	-50	0	0	0
3.999	40	-50	0	0	0
4	-10	-20	0	0	0
4.999	-10	-20	0	0	0
5	0	0	1	20	30
5.999	0	0	1	20	30
6	0	0	1	-30	20
6.999	0	0	1	-30	20
7	0	0	1	40	-50
7.999	0	0	1	40	-50
8	0	0	0	-10	-20
8.999	0	0	0	-10	-20

6. From this above sheet we are verifying the AND, OR and SWITCH functionality.
   - While testing AND -> a, b, c(control input 1) give custom inputs, make d, e as 0.



time	a	b	c	d	e
1	20	30	1	0	0
1.999	20	30	1	0	0
2	-30	20	0	0	0
2.999	-30	20	0	0	0
3	40	-50	0	0	0
3.999	40	-50	0	0	0
4	-10	-20	0	0	0
4.999	-10	-20	0	0	0

- While testing OR -> give custom inputs to d, e others make 0.

5	0	0	1	20	30
5.999	0	0	1	20	30
6	0	0	1	-30	20
6.999	0	0	1	-30	20
7	0	0	1	40	-50
7.999	0	0	1	40	-50
8	0	0	0	-10	-20
8.999	0	0	0	-10	-20

7. Go to Modelling
   - Change Model settings as

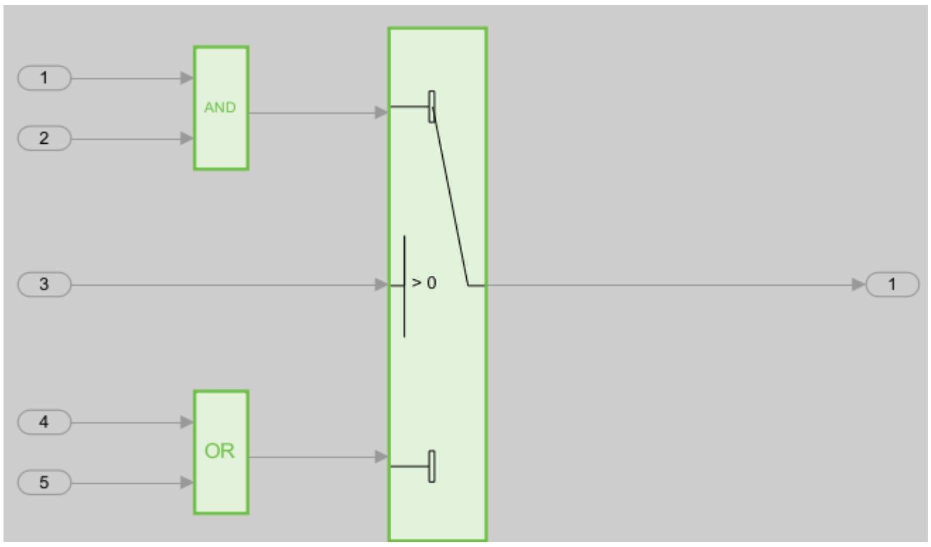
     Solver -> Type : Fixed step , dicrete(no continuous states)

     Fixed step size : 0.01 then ok.
   - Code generation -> system target file : ert.tlc(Embedded coder) then ok.
   - Coverage -> Click on Enable coverage analysis ->Entire system
     Coverage metrices -> Structural coverage level : Condition decision then ok and apply.

   To check whether settings are correct on run button we get a small popup in combination of pink and blue.

8. Click on Run after if u get the entire block in green this indicates the correct functionality test by custom inputs.

   Here we get a coverage report for the file which includes the information like Analysis information , coverage data information, model information, simulation optimization options, aggregated tests, model summary related to blocks used in the program.



9. Generate code using Embedded coder -> build code -> fix if error occurs and rebuild.

❖ Embedded Code :

```
#include "task1.h"
#include "rtwtypes.h"
#include "task1_private.h"
/* Block states (default storage) */
DW_task1_T task1_DW;
/* External outputs (root outports fed by signals with default storage) */
ExtY_task1_T task1_Y;
/* Real-time model */
static RT_MODEL_task1_T task1_M_;
RT_MODEL_task1_T *const task1_M = &task1_M_;
/* Model step function */
void task1_step(void)
{
  /* local block i/o variables */
  real_T rtb_FromWs[5];
  /* FromWorkspace: '<S1>/FromWs' */
  {
    real_T *pDataValues = (real_T *) task1_DW.FromWs_PWORK.DataPtr;
    real_T *pTimeValues = (real_T *) task1_DW.FromWs_PWORK.TimePtr;
    int_T currTimeIndex = task1_DW.FromWs_IWORK.PrevIndex;
    real_T t = task1_M->Timing.t[0];
   /* Get index */
    if (t <= pTimeValues[0]) {
      currTimeIndex = 0;
    } else if (t >= pTimeValues[29]) {
      currTimeIndex = 28;
    } else {
      if (t < pTimeValues[currTimeIndex]) {
        while (t < pTimeValues[currTimeIndex]) {
          currTimeIndex--;
        }
      } else {
        while (t >= pTimeValues[currTimeIndex + 1]) {
          currTimeIndex++;
        }
      }
    }
    task1_DW.FromWs_IWORK.PrevIndex = currTimeIndex;
    /* Post output */
    {
      real_T t1 = pTimeValues[currTimeIndex];
      real_T t2 = pTimeValues[currTimeIndex + 1];
      if (t1 == t2) {
        if (t < t1) {
       {
          int_T elIdx;
          for (elIdx = 0; elIdx < 5; ++elIdx) {
            (&rtb_FromWs[0])[elIdx] = pDataValues[currTimeIndex];
          pDataValues += 30;
        }
      }
    } else {
      {
        int_T elIdx;
        for (elIdx = 0; elIdx < 5; ++elIdx) {
          (&rtb_FromWs[0])[elIdx] = pDataValues[currTimeIndex + 1];
          pDataValues += 30;
        }
      }
    }
  } else {
    real_T f1 = (t2 - t) / (t2 - t1);
    real_T f2 = 1.0 - f1;
    real_T d1;
    real_T d2;
    int_T TimeIndex = currTimeIndex;
  {
    int_T elIdx;
    for (elIdx = 0; elIdx < 5; ++elIdx) {
      d1 = pDataValues[TimeIndex];
     d2 = pDataValues[TimeIndex + 1];
      (&rtb_FromWs[0])[elIdx] = (real_T) rtInterpolate(d1, d2, f1, f2);

      pDataValues += 30;

    }

   }

  }

 }

}
/* Switch: '<S2>/Switch' */
if (rtb_FromWs[2] > 0.0) {
 /* Outport: '<Root>/Out1' incorporates:
  *  Logic: '<S2>/Logical Operator'
  */
 task1_Y.Out1 = ((rtb_FromWs[0] != 0.0) && (rtb_FromWs[1] != 0.0));
} else {
 /* Outport: '<Root>/Out1' incorporates:
  *  Logic: '<S2>/Logical Operator1'
  */
 task1_Y.Out1 = ((rtb_FromWs[3] != 0.0) || (rtb_FromWs[4] != 0.0));
}
/* End of Switch: '<S2>/Switch' */
```

```
  /* Update absolute time for base rate */
  /* The "clockTick0" counts the number of times the code of this task has
   * been executed. The absolute time is the multiplication of "clockTick0"
   * and "Timing.stepSize0". Size of "clockTick0" ensures timer will not
   * overflow during the application lifespan selected.
   */
  task1_M->Timing.t[0] =
   ((time_T)(++task1_M->Timing.clockTick0)) * task1_M->Timing.stepSize0;
  {
   /* Update absolute timer for sample time: [0.01s, 0.0s] */
   /* The "clockTick1" counts the number of times the code of this task has
    * been executed. The resolution of this integer timer is 0.01, which is the step siz
    * of the task. Size of "clockTick1" ensures timer will not overflow during the
    * application lifespan selected.
    */
   task1_M->Timing.clockTick1++;
  }
}
/* Model initialize function */
void task1_initialize(void)
{
  /* Registration code */
  {
   /* Setup solver object */
   rtsiSetSimTimeStepPtr(&task1_M->solverInfo, &task1_M->Timing.simTimeStep);
   rtsiSetTPtr(&task1_M->solverInfo, &rtmGetTPtr(task1_M));
   rtsiSetStepSizePtr(&task1_M->solverInfo, &task1_M->Timing.stepSize0);
   rtsiSetErrorStatusPtr(&task1_M->solverInfo, (&rtmGetErrorStatus(task1_M)));
   rtsiSetRTModelPtr(&task1_M->solverInfo, task1_M);
  }
  rtsiSetSimTimeStep(&task1_M->solverInfo, MAJOR_TIME_STEP);
  rtsiSetSolverName(&task1_M->solverInfo,"FixedStepDiscrete");
  rtmSetTPtr(task1_M, &task1_M->Timing.tArray[0]);
  task1_M->Timing.stepSize0 = 0.01;
  /* Start for FromWorkspace: '<S1>/FromWs' */
  {
   static real_T pTimeValues0[] = { 1.0, 1.999, 1.999, 2.0, 2.0, 2.999, 2.999,
    3.0, 3.0, 3.999, 3.999, 4.0, 4.0, 4.999, 4.999, 5.0, 5.0, 5.999, 5.999,
    6.0, 6.0, 6.999, 6.999, 7.0, 7.0, 7.999, 7.999, 8.0, 8.0, 8.999 } ;
   static real_T pDataValues0[] = { 20.0, 20.0, 20.0, -30.0, -30.0, -30.0,
    -30.0, 40.0, 40.0, 40.0, 40.0, -10.0, -10.0, -10.0, -10.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 30.0, 30.0,
    30.0, 20.0, 20.0, 20.0, 20.0, -50.0, -50.0, -50.0, -50.0, -20.0, -20.0,
    -20.0, -20.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 20.0, 20.0, 20.0, 20.0, -30.0, -30.0, -30.0, -30.0, 40.0,
    40.0, 40.0, 40.0, -10.0, -10.0, -10.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
    0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 30.0, 30.0, 30.0, 30.0, 20.0, 20.0,
    20.0, 20.0, -50.0, -50.0, -50.0, -50.0, -20.0, -20.0, -20.0 }
   task1_DW.FromWs_PWORK.TimePtr = (void *) pTimeValues0;
   task1_DW.FromWs_PWORK.DataPtr = (void *) pDataValues0;
   task1_DW.FromWs_IWORK.PrevIndex = 0;
  }
}
/* Model terminate function */
void task1_terminate(void)
{
  /* (no terminate code required) */
}
/*
 * File trailer for generated code.
 *
 * [EOF]
 */
```

❖ SIL Testing :

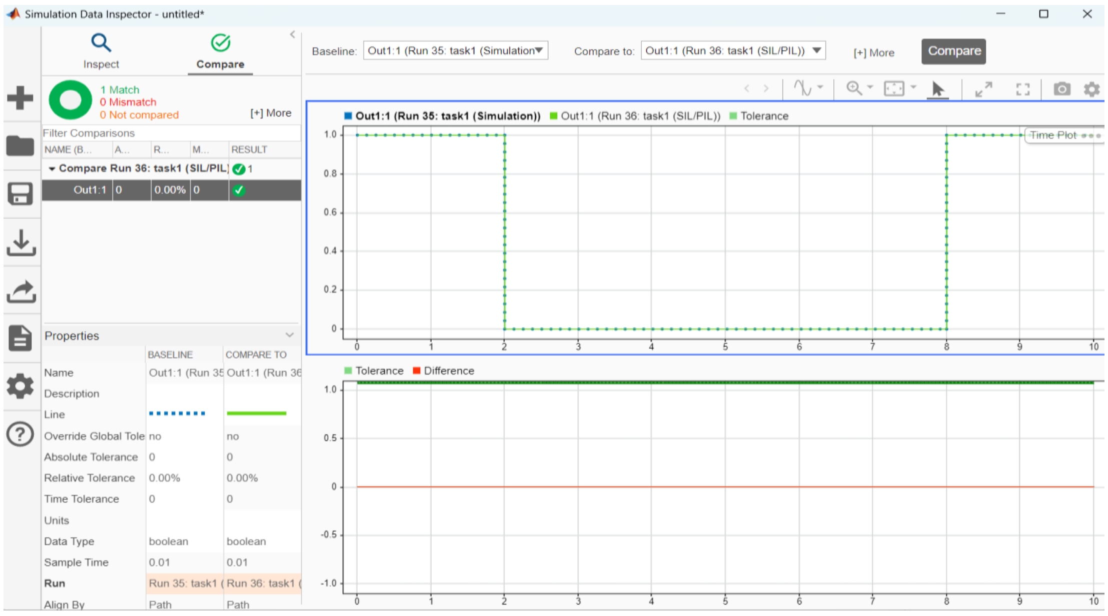
SIL (Software-in-the-Loop) testing in MATLAB/Simulink is used to verify the **generated code** by running it in a simulated environment. This ensures that the **auto-generated C/C++ code** behaves exactly like the Simulink model before deploying it to hardware.

Used for Code Verification (Ensures generated code produces expected results), Early Bug Detection (Identifies issues before deployment), Performance Analysis

(Measures execution time and efficiency), Consistency Check (Confirms that the model and generated code give the same outputs).

❖ <u>Procedure for SIL :</u>

1. Go to Apps.
2. Select SIL/PIL manager.
   Change System under test : Model block in SIL PIL mode.
3. Click on Run Verification.
4. Check the graph and right click and customize with dotted lines.
5. If the dots are on same path the verification and validation is correct.



Simulation Data Inspector - untitled*
Inspect
Compare
1 Match
0 Mismatch
0 Not compared
[+] More
Filter Comparisons
Baseline: Out1:1 (Run 35: task1 (Simulation)
Compare to: Out1:1 (Run 36: task1 (SIL/PIL))
[+] More
Compare
Out1:1 (Run 35: task1 (Simulation))
Out1:1 (Run 36: task1 (SIL/PIL))
Tolerance
Time Plot
Tolerance
Difference

NAME (B...	A...	R...	M...	RESULT
Compare Run 36: task1 (SIL/PIL)				✓ 1
Out1:1	0	0.00%	0	✓

Properties

	BASELINE	COMPARE TO
Name	Out1:1 (Run 35	Out1:1 (Run 36
Description		
Line	········	━━━
Override Global Tole	no	no
Absolute Tolerance	0	0
Relative Tolerance	0.00%	0.00%
Time Tolerance	0	0
Units		
Data Type	boolean	boolean
Sample Time	0.01	0.01
Run	Run 35: task1 (	Run 36: task1 (
Align By	Path	Path