



---

# *SmartVehicle* Evaluation Report

---

Code Generation

---

In support of the University of California Open  
Experimental Platform for DARPA – MoBIES,  
Contract F33615-01-C-1841.

Contact:  
William P. Milam  
Ford Research Laboratory  
Dearborn, MI  
[wmilam@ford.com](mailto:wmilam@ford.com)

Trademark and copyright Notices:

- TargetLink is a branded product of dSPACE Inc.
- The terms MATLAB, Simulink, Stateflow, and Real-Time Workshop, are all registered trademarks of The MathWorks, Inc. Embedded Coder is a brand of The MathWorks, Inc.
- Other product or brand names are trademarks of their respective holders.

Conventions used within this document:

- Informational messages (visible on command line, pop-up windows etc.) and input command messages (user input strings, selection items, etc.) will be indicated by `courier font`.
- **Bold font** will indicate Simulink, Stateflow, and Targetlink block names, signal names, system, and subsystem names.
- Parameter values, variables, and constant values will be indicated by *an italicized font*.



## 1. Overview

In response to the Embedded Software Analysis and Generation challenge problems, two code generation tools were developed by MoBIES Phase One participants: FORGES from Kestrel and ECSL from Vanderbilt. In this report we will evaluate these tools against the stated challenge problems and compare them to the commercial tools that were examined as part of the Baseline evaluation report.

## 2. Model Selection

Of the two tools developed, one was only able to support a subset of the Stateflow language. For that reason it was decided to apply the tools to a section of the transmission control model used in the baseline report. While this model does contain some Simulink blocks, they are table look up and gains. The bulk of the behavior of the model is contained in the shift\_schedule statemachine.

For each tool, code will be generated from this model. We will then review the completeness and accuracy of the code generated, it's equivalent behavior versus the model, code size and readability. All of these attributes were covered in the challenge problem document. We will also look at how the baseline tools address the same model.

## 3. Methodology

In order to ensure a complete analysis, we established a set of methods for evaluation of the code generators. The context of the experiment is that the models represent subsystems for which code is to be generated and eventually integrated into a larger software structure. The report is organized as follows:

The Shift\_Schedule model is described for both the FORGES and ECSL tools. The required model modifications for successful code generation are detailed. The discussion covers both simple and complex tool settings. The intent is to give the reader a feel for some of the issues involved with the basic operation of the tools. Additionally, we discuss some of the requirements for controlling the output of the tool and managing the characteristics of the generated C code. The models and generated code are then compared in simulation, and the tool capabilities are tabulated with respect to the challenge problem requirements. The generated code is included in the appendices.

The data used to verify the behavior of the model were derived from a test vector generation tool developed by CMU. It should be noted that the verification test data represent a single test vector. No attempt was made to exhaustively test the generated code. An assessment of correctness for the generated code is made exclusively by the comparison of simulation results for the model and code over the test cycle.

## 4. Tools

### 4.1. FORGES

The Kestrel Research Institute has been working on a generator of generators approach to the automatic code generation problem. They developed a formal semantic description of a subset of the Stateflow language and a formal semantic description of the C language. From these formal descriptions they generate a translation mechanism that will create C code from the input language, which is Stateflow in this instance, that is correct by construction. Correct by construction implies that the resulting code is a formally derived translation from Stateflow to C. There are several publications from Kestrel that better explain the correct by construction concept. [1][2]

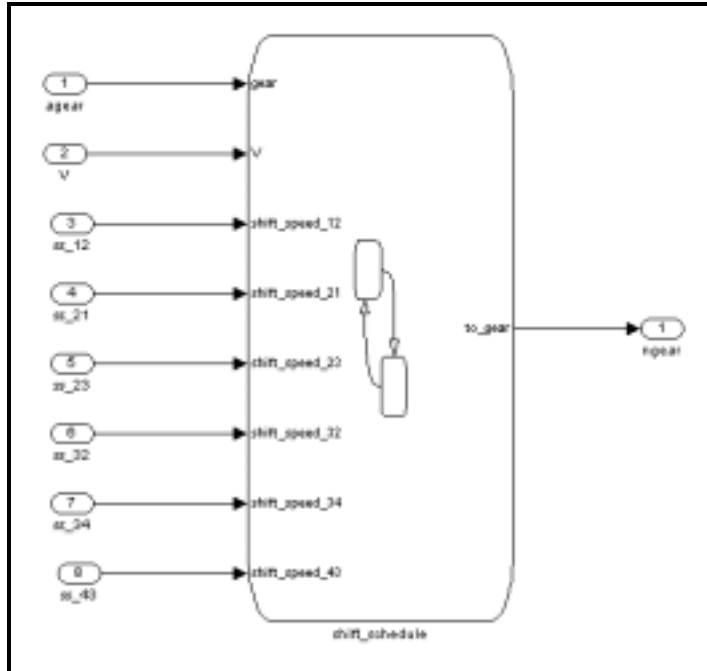
### 4.2. ECSL – Embedded Control Systems Modeling Language

Vanderbilt University has taken a similar approach in that they generate the code generator. They, however, use a meta-model of the Matlab Simulink/Stateflow language. This approach does not have the proof of correctness property associated with the FORGES tool. The ECSL tool does support a larger subset of Stateflow and discrete Simulink languages. In order to better understand the Metamodel approach there are several papers available from the Vanderbilt ISIS website. [3][4][5]

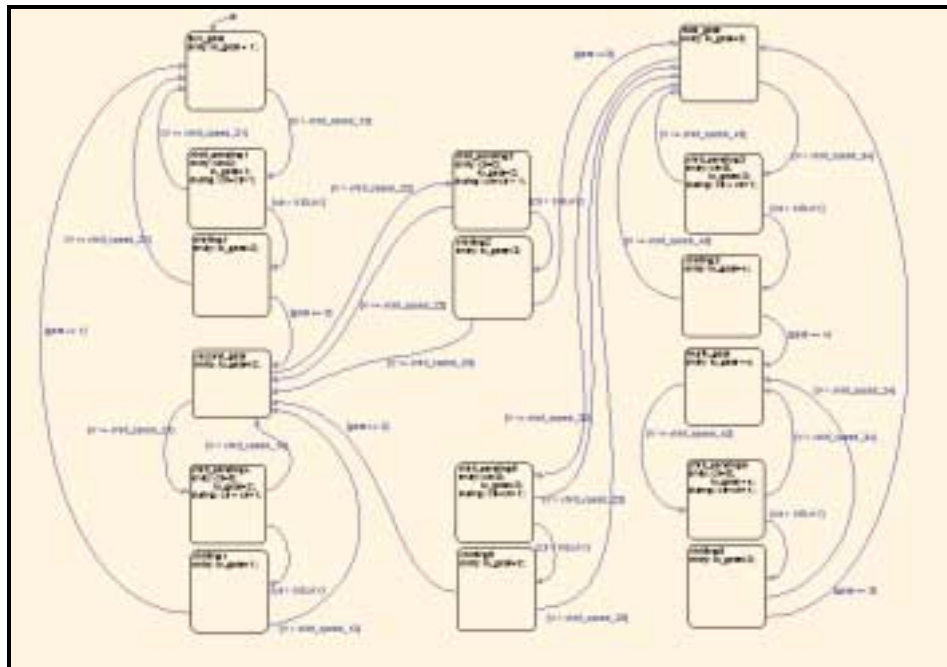
## 5. Results

### 5.1. The Model

As was stated earlier, we used a Stateflow only model which was derived from the Transmission Control sub-system as developed by UCB. The top level Stateflow representation is shown in Figure 1. As you can see we have eight inputs and one output. In addition there is one constant, DELAY, which is defined in the Stateflow data dictionary as a Matlab workspace value. As neither tool supports the use of workspace variables we opted to make it a constant. Another modification from the original was to flatten the state machine by promoting the sub-states to top level states. This required the addition of some transitions to preserve the machine behavior. While the use of hierarchical state machines is desired, for this example they were more of a syntactic convenience for the modeler. The resulting flat state machine is seen in Figure 2.



• Figure 1 Stateflow top level model



• Figure 2 Stateflow model showing states and transitions

## 5.2. FORGES

The FORGES tool is strictly a proof of concept. It is a demonstration of the broader concept of generating provably correct generators. With this in mind we were pleasantly surprised with the code generated by the tool.

The tool has two modes for usage: local and web based. The local version required a run-time license for a LISP program that we did not have installed. The web version was running on a server at Kestrel and proved easy to use, although somewhat problematic. The first model we used as a simple test was a two state model which simply oscillated between states. When we submitted this to the web server the code was generated quickly and was easy to compile and test. When, however we submitted the Shift Control state machine we seemed to go off into an endless loop. This was due to the time FORGES took to generate the code, approximately 14 minutes, during which time our firewall at Ford timed out and we lost connection to the web device. A copy of the source code is in the appendix.

Once we had the code, we built a Simulink S-Function to allow us to run the code versus the original model in a test harness. FORGES assumed a global shared value approach to passing inputs and outputs. This fairly closely follows the actual code practice used in powertrain controllers and also made it fairly easy to create the wrapper S-Function.

When we had the harness complete, we ran our first test and found that the code did indeed replicate the model behavior. When we re-ran the test, however, there was no agreement between code and model. This was traced to the FORGES assumption that the code need only be initialized once. Therefore, all of the default values used to track the states and transitions were set for initialization only when the code was loaded. Any subsequent run would use the last values stored. Because the FORGES code was so easy to read and comprehend we found it trivial to add an explicit initialization function which would be called each time the model was run. This routine is shown in the source code as `chart_init()`.

## 5.3. ECSL

The ECSL tool from Vanderbilt translates the Matlab Simulink/Stateflow models into an XML form. This is then translated into an ECSL model. To use ECSL the user needs a copy of GME2000, available from Vanderbilt. GME2000 is a graphical meta modeling environment which is described in various publications from Vanderbilt. For our purposes it suffices to say that GME2000 provides a user interface to allow the user to select which portions of the model for which they wish to generate code. So, for example, if you have a model with several sub-systems you can select particular sub-systems to code. In our case, there is only one member of the model, the state machine. However when the Stateflow block in Figure 1 is selected, ECSL would only generate what was essentially a function prototype. When we sent the model to Vanderbilt, they were able to generate code and send it back to us. At this time we do not know why this happened or what a workaround might be.

We repeated the same procedure as in the FORGES case. First, create a wrapper S-Function, add it to a test harness model and then run test vectors. Just like the FORGES code, we also found that the ECSL code did not seem to reset for subsequent runs, although when we examined the code we found a routine that seemed to reset the machine. However, when we tried to use it for that purpose we detected no change in behavior.

For the FORGES code we found it easy to navigate and understand. The ECSL code is an entirely different matter. Where FORGES was able to use two source files, one for source code and one header file, ECSL required seven files. Of the seven files, five were common files required to support Stateflow. The other two files were unique to the Shift Schedule model and were both C source files. Trying to decipher the two files specific to the model was impossible without the support files, as much of the state behavior is realized as a tree data structure. While the idea of using a tree structure may have some advantages for very large state machines, it results in code that is difficult to read and understand. This large source code problem was also apparent when we compiled the code using a production PowerPC compiler.

## 6. Comparison

In order to assess how well the tools performed against the current baseline, we also used the TargetLink and RTW-Embedded Coder tools to generate code. As we did for the Kestrel and Vanderbilt tools, we also created S-Functions and added them to the test harness. To make it easy to compare the results we offer the following table.

Item	Criteria	FORGES	ECSL	TL	RTW-EC
1	<b>ANSI C Compliance</b>	Yes	No	Yes	Yes
2	<b>Preserved Function (SL)</b>	N/A	N/A	N/A	N/A
3	<b>Preserved Function (SF)</b>	Yes	Yes	Yes	Yes
4	<b>ROM/RAM</b>	2508	7340	1596	1620
5	<b>Traceable/Readable</b>	Yes	No	Yes	No
6	<b>Variable Initialization</b>	No	No	No	No
7	<b>Number of files generated</b>	2	2	2	9
8	<b>Number of library files</b>	0	5	1	numerous

Notes:

- 1) ECSL used some C++ syntax in the library routines
- 6) Because it was easy to read and comprehend the FORGES and Targetlink code it was easy to add routines to handle initialization. This was not resolved for either ECSL or RTW-Embedded Coder.

## 7. Conclusion

The surprising result for us and Kestrel was the quality and size of the code generated. It has taken both dSpace and the MathWorks many years to develop their respective code generation tools. Kestrel took less than two years. In addition, because it is based on an analytical approach to generating the code generator, it is relatively easy to extend the supported Stateflow language and create a new code generator. We believe this approach is extremely promising and hope that commercial tool vendors will take notice.

The Vanderbilt approach also is about generating generators, however their approach yielded results that were not as efficient as the FORGES tool. They did generate code quickly, recall it took 14 minutes for FORGES to produce code, but the benefits seem obvious. That said, the Vanderbilt tools have produced working code. In addition they have demonstrated the versatility of their GME 2000 tool and meta modeling in other applications in the MoBIES program, most notably in the HSIF and Model Compiler tools. Their ability to quickly synthesize tool environments and map languages is impressive.

## 8. Bibliography

[1] Anton, J., McDonald, J., "SPECWARE: Producing Software Correct by Construction", Kestrel Technical Report, March 14, 2001

[2] Smith, D., "Mechanizing the Development of Software", *Calculational System Design* Proceedings of the International Summer School Marktoberdorf, Ed. M. Broy, NATO ASI Series, IOS Press, Amsterdam, 1999

[3] Nordstrom, G., "Formalizing the Specification of Graphical Modeling Languages", Proceedings of the IEEE Aerospace 2000 Conference, CD-ROM Reference 10.0402, Big Sky, MT, March 2000

[4] Ledeczki, A., Bakay, A., Maroti, M., "Model-Integrated Embedded Systems", in Robertson, Shrobe, Laddaga (Ed), *Self Adaptive Software*, Springer-Verlag Lecture Notes in CS, #1936, February, 2001

[5] <http://www.isis.vanderbilt.edu> : Home page for the Institute for Software Integrated Systems.

## 9. Appendices

### 9.1. FORGES Code

#### 9.1.1. forges1a.h

```
/* External Variables */
```

```

extern int gear;
extern double V;
extern double shift_speed_12;
extern double shift_speed_21;
extern double shift_speed_23;
extern double shift_speed_32;
extern double shift_speed_34;
extern double shift_speed_43;
extern int to_gear;
/*
 * Input Events
 */

/*
 * Output Events
 */

```

### 9.1.2. forges1a.c

```

/* Include files */

#include "forges1a.h"

/* Definitions */

/* Variables */
static int event;
static int test;
static int ctr;
int DELAY = 1;

/* Functions */
void chart_shift_schedule_2 ();
void chart_init ();

/* Variable definitions */
static int active_state_third_gear_3;
static int active_state_first_gear_4;
static int active_state_shifting6_5;
static int active_state_shifting5_6;
static int active_state_shifting4_7;
static int active_state_shift_pending1_8;
static int active_state_shift_pending2_9;
static int active_state_shift_pending3_10;
static int active_state_shifting1_11;
static int active_state_shifting2_12;
static int active_state_shifting3_13;
static int active_state_fourth_gear_14;
static int active_state_second_gear_15;
static int active_state_shift_pending6_16;
static int active_state_shift_pending5_17;
static int active_state_shift_pending4_18;
static int active_chart_shift_schedule_2;

void chart_init() /*This routine added by Ford*/
{

```

```
active_state_third_gear_3 = 0;
active_state_first_gear_4 = 0;
```

```
active_state_shifting6_5 = 0;
active_state_shifting5_6 = 0;
active_state_shifting4_7 = 0;
active_state_shift_pending1_8 = 0;
active_state_shift_pending2_9 = 0;
active_state_shift_pending3_10 = 0;
active_state_shifting1_11 = 0;
active_state_shifting2_12 = 0;
active_state_shifting3_13 = 0;
active_state_fourth_gear_14 = 0;
active_state_second_gear_15 = 0;
active_state_shift_pending6_16 = 0;
active_state_shift_pending5_17 = 0;
active_state_shift_pending4_18 = 0;
active_chart_shift_schedule_2 = 0;
}
```

```
/* Function definitions */
```

```
void chart_shift_schedule_2 () {
  if (active_chart_shift_schedule_2) {
    if (active_state_shift_pending4_18) {
      test = (V > shift_speed_12);
      if (test) {
        active_state_shift_pending4_18 = 0;
        active_state_second_gear_15 = 1;
        to_gear = 2;
        if (!test) {
          ctr = (ctr + 1);
        }
      } else {
        test = (ctr > DELAY);
        if (test) {
          active_state_shift_pending4_18 = 0;
          active_state_shifting4_7 = 1;
          to_gear = 1;
        }
        if (!test) {
          ctr = (ctr + 1);
        }
      }
    } else {
      if (active_state_shift_pending5_17) {
        test = (V > shift_speed_23);
        if (test) {
          active_state_shift_pending5_17 = 0;
          active_state_third_gear_3 = 1;
          to_gear = 3;
          if (!test) {
            ctr = (ctr + 1);
          }
        } else {
          test = (ctr > DELAY);
          if (test) {
            active_state_shift_pending5_17 = 0;
            active_state_shifting5_6 = 1;
            to_gear = 2;
          }
        }
      }
    }
  }
}
```

```
if (!test) {  
    ctr = (ctr + 1);  
}
```

```
}
```

```
}  
} else {  
    if (active_state_shift_pending6_16) {  
        test = (ctr > DELAY);  
        if (test) {  
            active_state_shift_pending6_16 = 0;  
            active_state_shifting6_5 = 1;  
            to_gear = 3;  
            if (!test) {  
                ctr = (ctr + 1);  
            }  
        } else {  
            test = (V > shift_speed_34);  
            if (!test) {  
                active_state_shift_pending6_16 = 0;  
                active_state_fourth_gear_14 = 1;  
                to_gear = 4;  
            }  
            if (!test) {  
                ctr = (ctr + 1);  
            }  
        }  
    } else {  
        if (active_state_second_gear_15) {  
            test = (V <= shift_speed_21);  
            if (test) {  
                active_state_second_gear_15 = 0;  
                active_state_shift_pending4_18 = 1;  
                ctr = 0;  
                to_gear = 2;  
            } else {  
                test = (V > shift_speed_23);  
                if (test) {  
                    active_state_second_gear_15 = 0;  
                    active_state_shift_pending2_9 = 1;  
                    ctr = 0;  
                    to_gear = 2;  
                }  
            }  
        } else {  
            if (active_state_fourth_gear_14) {  
                test = (V <= shift_speed_43);  
                if (test) {  
                    active_state_fourth_gear_14 = 0;  
                    active_state_shift_pending6_16 = 1;  
                    ctr = 0;  
                    to_gear = 4;  
                }  
            } else {  
                if (active_state_shifting3_13) {  
                    test = (V <= shift_speed_43);  
                    if (test) {  
                        active_state_shifting3_13 = 0;  
                        active_state_third_gear_3 = 1;  
                        to_gear = 3;  
                    } else {
```

```

if (test) {
    active_state_shifting3_13 = 0;
    active_state_fourth_gear_14 = 1;
    to_gear = 4;

```

```

}

```

```

}
} else {
    if (active_state_shifting2_12) {
        test = (V <= shift_speed_23);
        if (test) {
            active_state_shifting2_12 = 0;
            active_state_second_gear_15 = 1;
            to_gear = 2;
        } else {
            test = (gear == 3);
            if (test) {
                active_state_shifting2_12 = 0;
                active_state_third_gear_3 = 1;
                to_gear = 3;
            }
        }
    } else {
        if (active_state_shifting1_11) {
            test = (V <= shift_speed_21);
            if (test) {
                active_state_shifting1_11 = 0;
                active_state_first_gear_4 = 1;
                to_gear = 1;
            } else {
                test = (gear == 2);
                if (test) {
                    active_state_shifting1_11 = 0;
                    active_state_second_gear_15 = 1;
                    to_gear = 2;
                }
            }
        } else {
            if (active_state_shift_pending3_10) {
                test = (ctr > DELAY);
                if (test) {
                    active_state_shift_pending3_10 = 0;
                    active_state_shifting3_13 = 1;
                    to_gear = 4;
                    if (!test) {
                        ctr = (ctr + 1);
                    }
                } else {
                    test = (V <= shift_speed_43);
                    if (test) {
                        active_state_shift_pending3_10 = 0;
                        active_state_third_gear_3 = 1;
                        to_gear = 3;
                    }
                    if (!test) {
                        ctr = (ctr + 1);
                    }
                }
            } else {
                if (active_state_shift_pending2_9) {

```

```

if (test) {
  active_state_shift_pending2_9 = 0;
  active_state_second_gear_15 = 1;
  to_gear = 2;
  if (!test) {
    ctr = (ctr + 1);
  }
}

```

```

} else {
  test = (ctr > DELAY);
  if (test) {
    active_state_shift_pending2_9 = 0;
    active_state_shifting2_12 = 1;
    to_gear = 3;
  }
  if (!test) {
    ctr = (ctr + 1);
  }
}
} else {
  if (active_state_shift_pending1_8) {
    test = (ctr > DELAY);
    if (test) {
      active_state_shift_pending1_8 = 0;
      active_state_shifting1_11 = 1;
      to_gear = 2;
      if (!test) {
        ctr = (ctr + 1);
      }
    }
  } else {
    test = (V <= shift_speed_21);
    if (test) {
      active_state_shift_pending1_8 = 0;
      active_state_first_gear_4 = 1;
      to_gear = 1;
    }
    if (!test) {
      ctr = (ctr + 1);
    }
  }
} else {
  if (active_state_shifting4_7) {
    test = (V > shift_speed_12);
    if (test) {
      active_state_shifting4_7 = 0;
      active_state_second_gear_15 = 1;
      to_gear = 2;
    } else {
      test = (gear == 1);
      if (test) {
        active_state_shifting4_7 = 0;
        active_state_first_gear_4 = 1;
        to_gear = 1;
      }
    }
  }
} else {
  if (active_state_shifting5_6) {
    test = (gear == 2);
    if (test) {
      active_state_shifting5_6 = 0;
    }
  }
}

```



```

    }
  }
}
}
}
} else {
  active_chart_shift_schedule_2 = 1;
  active_state_first_gear_4 = 1;
  to_gear = 1;

```

```

}

```

```

}

```

## 9.2. ECSL Code

### 9.2.1. Shift Schedule SL.c

```

/* shift_schedule_SL.c generated on 17:46:27 Wednesday, June 19, 2002 */

#include "sf.h"
#include "ws.h"

extern void C1_sf_init(sf_context *);
extern void C1_sf_input(sf_context *);
extern void C1_sf_output(sf_context *);
sf_context C1_sf_ctx;
bool C1_sf_ctx_init=false;

void shift_schedule1_init(void)
{
  C1_sf_ctx_init = false;
}

/*
 * Block: shift_schedule
 * gear -> in_0
 * V -> in_1
 * shift_speed_12 -> in_2
 * shift_speed_21 -> in_3
 * shift_speed_23 -> in_4
 * shift_speed_32 -> in_5
 * shift_speed_34 -> in_6
 * shift_speed_43 -> in_7
 * to_gear -> out_0
 */
void shift_schedule1 (
  double in_0, double in_1, double in_2, double in_3, double in_4, double in_5, double in_6, double in_7,
  double *out_0
)
{
  double l_0 = 0;
  double l_1 = 0;
  double l_2 = 0;
  double l_3 = 0;

```

```

double L_5 = 0;
double L_6 = 0;
double L_7 = 0;
double L_8 = 0;
double L_9 = 0;

```

```

L_0 = in_0;
L_1 = in_1;
L_2 = in_2;
L_3 = in_3;
L_4 = in_4;
L_5 = in_5;

```

```

L_6 = in_6;

```

```

L_7 = in_7;

```

```

if (!C1_sf_ctx_init) { C1_sf_init(&C1_sf_ctx); C1_sf_ctx_init=true; }
C1_sf_ctx.inputs[0] = L_0;
C1_sf_ctx.inputs[1] = L_1;
C1_sf_ctx.inputs[2] = L_2;
C1_sf_ctx.inputs[3] = L_3;
C1_sf_ctx.inputs[4] = L_4;
C1_sf_ctx.inputs[5] = L_5;
C1_sf_ctx.inputs[6] = L_6;
C1_sf_ctx.inputs[7] = L_7;
C1_sf_input(&C1_sf_ctx);
sf_step(&C1_sf_ctx);
C1_sf_output(&C1_sf_ctx);
L_8 = C1_sf_ctx.outputs[0];
L_9 = C1_sf_ctx.outputs[1];

*out_0 = L_9;
}

```

## 9.2.2. Shift Schedule SF.c

```

/* shift_schedule_SF.c generated on 17:46:27 Wednesday, June 19, 2002 */

#include "sf.h"

/*
/* create all the expressions
*/
bool C1_t_guard_0(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv <= (context->data[6] /*shift_speed_21*/).data.dv);
    return ret;
}
bool C1_t_guard_1(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv <= (context->data[2] /*shift_speed_32*/).data.dv);
    return ret;
}

```

```

bool C1_t_guard_2(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv <= (context->data[0] /*shift_speed_43*/).data.dv);
    return ret;
}
bool C1_t_guard_3(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[7] /*shift_speed_12*/).data.dv);
    return ret;
}
bool C1_t_guard_4(sf_context *context)
{
    bool ret = true

```

```

&& ((context->data[8] /*V*/).data.dv <= (context->data[3] /*shift_speed_23*/).data.dv);

```

```

    return ret;
}
bool C1_t_guard_5(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[7] /*shift_speed_12*/).data.dv);
    return ret;
}
bool C1_t_guard_6(sf_context *context)
{
    bool ret = true
    && ((context->data[9] /*gear*/).data.bv == 2);
    return ret;
}
bool C1_t_guard_7(sf_context *context)
{
    bool ret = true
    && ((context->data[9] /*gear*/).data.bv == 2);
    return ret;
}
bool C1_t_guard_8(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv <= (context->data[3] /*shift_speed_23*/).data.dv);
    return ret;
}
bool C1_t_guard_9(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[1] /*shift_speed_34*/).data.dv);
    return ret;
}
bool C1_t_guard_10(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[1] /*shift_speed_34*/).data.dv);
    return ret;
}
bool C1_t_guard_11(sf_context *context)
{
    bool ret = true
    && ((context->data[9] /*gear*/).data.bv == 4);
    return ret;
}

```

```

bool C1_t_guard_12(sf_context *context)
{
    bool ret = true
    && ((context->data[5] /*ctr*/).data.bv > (context->data[4] /*DELAY*/).data.bv);
    return ret;
}
bool C1_t_guard_13(sf_context *context)
{
    bool ret = true
    && ((context->data[5] /*ctr*/).data.bv > (context->data[4] /*DELAY*/).data.bv);
    return ret;
}
bool C1_t_guard_14(sf_context *context)
{
    bool ret = true
    && ((context->data[5] /*ctr*/).data.bv > (context->data[4] /*DELAY*/).data.bv);
    return ret;
}

```

```

}

```

```

bool C1_t_guard_15(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[1] /*shift_speed_34*/).data.dv);
    return ret;
}
bool C1_t_guard_16(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[3] /*shift_speed_23*/).data.dv);
    return ret;
}
bool C1_t_guard_17(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[7] /*shift_speed_12*/).data.dv);
    return ret;
}
bool C1_t_guard_18(sf_context *context)
{
    bool ret = true
    && ((context->data[5] /*ctr*/).data.bv > (context->data[4] /*DELAY*/).data.bv);
    return ret;
}
bool C1_t_guard_19(sf_context *context)
{
    bool ret = true
    && ((context->data[5] /*ctr*/).data.bv > (context->data[4] /*DELAY*/).data.bv);
    return ret;
}
bool C1_t_guard_20(sf_context *context)
{
    bool ret = true
    && ((context->data[5] /*ctr*/).data.bv > (context->data[4] /*DELAY*/).data.bv);
    return ret;
}
bool C1_t_guard_21(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv <= (context->data[6] /*shift_speed_21*/).data.dv);
    return ret;
}

```

```

bool C1_t_guard_22(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv <= (context->data[6] /*shift_speed_21*/).data.dv);
    return ret;
}
bool C1_t_guard_23(sf_context *context)
{
    bool ret = true
    && ((context->data[9] /*gear*/).data.bv == 1);
    return ret;
}
bool C1_t_guard_24(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[3] /*shift_speed_23*/).data.dv);
    return ret;
}
bool C1_t_guard_25(sf_context *context)

```

```
{
```

```

    bool ret = true
    && ((context->data[8] /*V*/).data.dv <= (context->data[0] /*shift_speed_43*/).data.dv);
    return ret;
}
bool C1_t_guard_26(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv <= (context->data[0] /*shift_speed_43*/).data.dv);
    return ret;
}
bool C1_t_guard_27(sf_context *context)
{
    bool ret = true
    && ((context->data[8] /*V*/).data.dv > (context->data[3] /*shift_speed_23*/).data.dv);
    return ret;
}
bool C1_t_guard_28(sf_context *context)
{
    bool ret = true
    && ((context->data[9] /*gear*/).data.bv == 3);
    return ret;
}
bool C1_t_guard_29(sf_context *context)
{
    bool ret = true
    && ((context->data[9] /*gear*/).data.bv == 3);
    return ret;
}
bool C1_s_entry_1(sf_context *context)
{
    (context->data[5] /*ctr*/).data.bv=0;
    (context->data[10] /*to_gear*/).data.bv=2;
    return true;
}
bool C1_s_during_1(sf_context *context)
{
    (context->data[5] /*ctr*/).data.bv=(context->data[5] /*ctr*/).data.bv + 1;
    return true;
}

```

```

{
  (context->data[5] /*ctr*/).data.bv=0;
  (context->data[10] /*to_gear*/).data.bv=3;
  return true;
}
bool C1_s_during_2(sf_context *context)
{
  (context->data[5] /*ctr*/).data.bv=(context->data[5] /*ctr*/).data.bv + 1;
  return true;
}
bool C1_s_entry_3(sf_context *context)
{
  (context->data[5] /*ctr*/).data.bv=0;
  (context->data[10] /*to_gear*/).data.bv=4;
  return true;
}
bool C1_s_during_3(sf_context *context)
{
  (context->data[5] /*ctr*/).data.bv=(context->data[5] /*ctr*/).data.bv + 1;
  return true;
}

```

```
bool C1_s_entry_4(sf_context *context)
```

```

{
  (context->data[10] /*to_gear*/).data.bv=2;
  return true;
}
bool C1_s_entry_5(sf_context *context)
{
  (context->data[10] /*to_gear*/).data.bv=4;
  return true;
}
bool C1_s_entry_6(sf_context *context)
{
  (context->data[10] /*to_gear*/).data.bv=4;
  return true;
}
bool C1_s_entry_7(sf_context *context)
{
  (context->data[10] /*to_gear*/).data.bv=3;
  return true;
}
bool C1_s_entry_8(sf_context *context)
{
  (context->data[10] /*to_gear*/).data.bv=2;
  return true;
}
bool C1_s_entry_9(sf_context *context)
{
  (context->data[5] /*ctr*/).data.bv=0;
  (context->data[10] /*to_gear*/).data.bv=3;
  return true;
}
bool C1_s_during_9(sf_context *context)
{
  (context->data[5] /*ctr*/).data.bv=(context->data[5] /*ctr*/).data.bv + 1;
  return true;
}
bool C1_s_entry_10(sf_context *context)
{

```

```

(context->data[10] /*to_gear*).data.bv=2;
return true;
}
bool C1_s_during_10(sf_context *context)
{
(context->data[5] /*ctr*).data.bv=(context->data[5] /*ctr*).data.bv + 1;
return true;
}
bool C1_s_entry_11(sf_context *context)
{
(context->data[5] /*ctr*).data.bv=0;
(context->data[10] /*to_gear*).data.bv=1;
return true;
}
bool C1_s_during_11(sf_context *context)
{
(context->data[5] /*ctr*).data.bv=(context->data[5] /*ctr*).data.bv + 1;
return true;
}
bool C1_s_entry_12(sf_context *context)
{
(context->data[10] /*to_gear*).data.bv=1;
return true;
}

```

```

}

```

```

bool C1_s_entry_13(sf_context *context)
{
(context->data[10] /*to_gear*).data.bv=2;
return true;
}
bool C1_s_entry_14(sf_context *context)
{
(context->data[10] /*to_gear*).data.bv=3;
return true;
}
bool C1_s_entry_15(sf_context *context)
{
(context->data[10] /*to_gear*).data.bv=1;
return true;
}
bool C1_s_entry_16(sf_context *context)
{
(context->data[10] /*to_gear*).data.bv=3;
return true;
}

void C1_sf_init(sf_context *context)
{
/* create states */
assert( 59 < MAX_STRINGS );
context->string_table[0] = "shift_schedule";
context->string_table[1] = "shift_pending4";
context->string_table[2] = "shift_pending5";
context->string_table[3] = "shift_pending6";
context->string_table[4] = "second_gear";
context->string_table[5] = "fourth_gear";
context->string_table[6] = "shifting3";
context->string_table[7] = "shifting2";
context->string_table[8] = "shifting1";
context->string_table[9] = "shift_pending3";
}

```

```

context->string_table[11] = "shift_pending1";
context->string_table[12] = "shifting4";
context->string_table[13] = "shifting5";
context->string_table[14] = "shifting6";
context->string_table[15] = "first_gear";
context->string_table[16] = "third_gear";
context->string_table[17] = "shift_speed_43";
context->string_table[18] = "shift_speed_34";
context->string_table[19] = "shift_speed_32";
context->string_table[20] = "shift_speed_23";
context->string_table[21] = "DELAY";
context->string_table[22] = "ctr";
context->string_table[23] = "shift_speed_21";
context->string_table[24] = "shift_speed_12";
context->string_table[25] = "V";
context->string_table[26] = "gear";
context->string_table[27] = "to_gear";
context->string_table[28] = "second_gear_shift_pending4_shift_schedule";
context->string_table[29] = "third_gear_shift_pending5_shift_schedule";
context->string_table[30] = "fourth_gear_shift_pending6_shift_schedule";
context->string_table[31] = "shift_pending4_second_gear_shift_schedule";
context->string_table[32] = "shifting2_second_gear_shift_schedule";
context->string_table[33] = "shifting4_second_gear_shift_schedule";
context->string_table[34] = "shifting5_second_gear_shift_schedule";
context->string_table[35] = "shifting1_second_gear_shift_schedule";

```

```

context->string_table[36] = "shift_pending2_second_gear_shift_schedule";

```

```

context->string_table[37] = "shifting6_fourth_gear_shift_schedule";
context->string_table[38] = "shift_pending6_fourth_gear_shift_schedule";
context->string_table[39] = "shifting3_fourth_gear_shift_schedule";
context->string_table[40] = "shift_pending3_shifting3_shift_schedule";
context->string_table[41] = "shift_pending2_shifting2_shift_schedule";
context->string_table[42] = "shift_pending1_shifting1_shift_schedule";
context->string_table[43] = "third_gear_shift_pending3_shift_schedule";
context->string_table[44] = "second_gear_shift_pending2_shift_schedule";
context->string_table[45] = "first_gear_shift_pending1_shift_schedule";
context->string_table[46] = "shift_pending4_shifting4_shift_schedule";
context->string_table[47] = "shift_pending5_shifting5_shift_schedule";
context->string_table[48] = "shift_pending6_shifting6_shift_schedule";
context->string_table[49] = "shifting1_first_gear_shift_schedule";
context->string_table[50] = "shift_pending1_first_gear_shift_schedule";
context->string_table[51] = "shifting4_first_gear_shift_schedule";
context->string_table[52] = "shift_pending5_third_gear_shift_schedule";
context->string_table[53] = "shifting3_third_gear_shift_schedule";
context->string_table[54] = "shift_pending3_third_gear_shift_schedule";
context->string_table[55] = "shifting5_third_gear_shift_schedule";
context->string_table[56] = "shifting6_third_gear_shift_schedule";
context->string_table[57] = "shifting2_third_gear_shift_schedule";

```

```

context->num_strings = 58;

```

```

context->num_states = 17;
assert( 17 < MAX_STATES );
context->root_state = 0;

```

```

context->num_events = 0;
assert( 0 < MAX_EVENTS );

```

```

context->num_data = 11;
assert( 11 < MAX_DATA );

```

```

context->num_transitions = 30;
assert( 30 < MAX_TRANSITIONS );

context->num_inputs = 8;
assert( 8 < MAX_INPUTS );
context->num_outputs = 1;
assert( 1 < MAX_OUTPUTS );
{
    sf_state * s;
    make_state(&context->states[0], 10000007, 0, sf_or);
    make_state(&context->states[1], 100000023, 1, sf_leaf);
    make_state(&context->states[2], 100000022, 2, sf_leaf);
    make_state(&context->states[3], 100000021, 3, sf_leaf);
    make_state(&context->states[4], 100000020, 4, sf_leaf);
    make_state(&context->states[5], 100000019, 5, sf_leaf);
    make_state(&context->states[6], 100000018, 6, sf_leaf);
    make_state(&context->states[7], 100000017, 7, sf_leaf);
    make_state(&context->states[8], 100000016, 8, sf_leaf);
    make_state(&context->states[9], 100000015, 9, sf_leaf);
    make_state(&context->states[10], 100000014, 10, sf_leaf);
    make_state(&context->states[11], 100000013, 11, sf_leaf);
    make_state(&context->states[12], 100000012, 12, sf_leaf);
    make_state(&context->states[13], 100000011, 13, sf_leaf);
    make_state(&context->states[14], 100000010, 14, sf_leaf);
    make_state(&context->states[15], 100000009, 15, sf_leaf);
    make_state(&context->states[16], 100000008, 16, sf_leaf);
}

```

```

/* reference the root state */

```

```

context->root_state = &context->states[0];
/* create state hierarchy */
{
    sf_state *p, *c;
    p = &context->states[ 0]; /* shift_schedule */
    make_sub_states(p, 16);
    c = &context->states[ 1]; set_parent(c, p); add_sub_state(p, c); /* shift_pending4 */
    c = &context->states[ 2]; set_parent(c, p); add_sub_state(p, c); /* shift_pending5 */
    c = &context->states[ 3]; set_parent(c, p); add_sub_state(p, c); /* shift_pending6 */
    c = &context->states[ 4]; set_parent(c, p); add_sub_state(p, c); /* second_gear */
    c = &context->states[ 5]; set_parent(c, p); add_sub_state(p, c); /* fourth_gear */
    c = &context->states[ 6]; set_parent(c, p); add_sub_state(p, c); /* shifting3 */
    c = &context->states[ 7]; set_parent(c, p); add_sub_state(p, c); /* shifting2 */
    c = &context->states[ 8]; set_parent(c, p); add_sub_state(p, c); /* shifting1 */
    c = &context->states[ 9]; set_parent(c, p); add_sub_state(p, c); /* shift_pending3 */
    c = &context->states[ 10]; set_parent(c, p); add_sub_state(p, c); /* shift_pending2 */
    c = &context->states[ 11]; set_parent(c, p); add_sub_state(p, c); /* shift_pending1 */
    c = &context->states[ 12]; set_parent(c, p); add_sub_state(p, c); /* shifting4 */
    c = &context->states[ 13]; set_parent(c, p); add_sub_state(p, c); /* shifting5 */
    c = &context->states[ 14]; set_parent(c, p); add_sub_state(p, c); /* shifting6 */
    c = &context->states[ 15]; set_parent(c, p); add_sub_state(p, c); /* first_gear */
    c = &context->states[ 16]; set_parent(c, p); add_sub_state(p, c); /* third_gear */
}
/* create default state */
{
    sf_state *p;
    sf_state *d;
    p = &context->states[ 0]; d = &context->states[ 15]; set_default_state(p, d);
}
/* create data */
{

```

```

d = make_double(&context->data[0], 200000048, 17, 0);
d = make_double(&context->data[1], 200000047, 18, 0);
d = make_double(&context->data[2], 200000046, 19, 0);
d = make_double(&context->data[3], 200000045, 20, 0);
d = make_char(&context->data[4], 200000044, 21, 3);
d = make_char(&context->data[5], 200000043, 22, 0);
d = make_double(&context->data[6], 200000042, 23, 1);
d = make_double(&context->data[7], 200000041, 24, 1);
d = make_double(&context->data[8], 200000040, 25, 1);
d = make_char(&context->data[9], 200000039, 26, 1);
d = make_char(&context->data[10], 200000038, 27, 1);
}
/* create events */
{
  sf_event * e;
}
/* create transitions */
{
  sf_transition * t;
  sf_state * o, * s, * d;
  o = &context->states[0]; /* shift_schedule */
  make_scope_transitions(o, 30);
  s = &context->states[4]; d = &context->states[1];
  t = make_transition(&context->transitions[0], 400000041, 28, o, s, d); add_scope_transition(o, t);
  s = &context->states[16]; d = &context->states[2];
  t = make_transition(&context->transitions[1], 400000024, 29, o, s, d); add_scope_transition(o, t);
  s = &context->states[5]; d = &context->states[3];
  t = make_transition(&context->transitions[2], 400000038, 30, o, s, d); add_scope_transition(o, t);
  s = &context->states[1]; d = &context->states[4];

```

```

t = make_transition(&context->transitions[3], 400000049, 31, o, s, d); add_scope_transition(o, t);

```

```

s = &context->states[7]; d = &context->states[4];
t = make_transition(&context->transitions[4], 400000047, 32, o, s, d); add_scope_transition(o, t);
s = &context->states[12]; d = &context->states[4];
t = make_transition(&context->transitions[5], 400000042, 33, o, s, d); add_scope_transition(o, t);
s = &context->states[13]; d = &context->states[4];
t = make_transition(&context->transitions[6], 400000040, 34, o, s, d); add_scope_transition(o, t);
s = &context->states[8]; d = &context->states[4];
t = make_transition(&context->transitions[7], 400000036, 35, o, s, d); add_scope_transition(o, t);
s = &context->states[10]; d = &context->states[4];
t = make_transition(&context->transitions[8], 400000035, 36, o, s, d); add_scope_transition(o, t);
s = &context->states[14]; d = &context->states[5];
t = make_transition(&context->transitions[9], 400000051, 37, o, s, d); add_scope_transition(o, t);
s = &context->states[3]; d = &context->states[5];
t = make_transition(&context->transitions[10], 400000039, 38, o, s, d); add_scope_transition(o, t);
s = &context->states[6]; d = &context->states[5];
t = make_transition(&context->transitions[11], 400000037, 39, o, s, d); add_scope_transition(o, t);
s = &context->states[9]; d = &context->states[6];
t = make_transition(&context->transitions[12], 400000034, 40, o, s, d); add_scope_transition(o, t);
s = &context->states[10]; d = &context->states[7];
t = make_transition(&context->transitions[13], 400000033, 41, o, s, d); add_scope_transition(o, t);
s = &context->states[11]; d = &context->states[8];
t = make_transition(&context->transitions[14], 400000032, 42, o, s, d); add_scope_transition(o, t);
s = &context->states[16]; d = &context->states[9];
t = make_transition(&context->transitions[15], 400000027, 43, o, s, d); add_scope_transition(o, t);
s = &context->states[4]; d = &context->states[10];
t = make_transition(&context->transitions[16], 400000031, 44, o, s, d); add_scope_transition(o, t);
s = &context->states[15]; d = &context->states[11];
t = make_transition(&context->transitions[17], 400000029, 45, o, s, d); add_scope_transition(o, t);
s = &context->states[1]; d = &context->states[12];

```

```

s = &context->states[2]; d = &context->states[13];
t = make_transition(&context->transitions[19], 400000044, 47, o, s, d); add_scope_transition(o, t);
s = &context->states[3]; d = &context->states[14];
t = make_transition(&context->transitions[20], 400000043, 48, o, s, d); add_scope_transition(o, t);
s = &context->states[8]; d = &context->states[15];
t = make_transition(&context->transitions[21], 400000048, 49, o, s, d); add_scope_transition(o, t);
s = &context->states[11]; d = &context->states[15];
t = make_transition(&context->transitions[22], 400000028, 50, o, s, d); add_scope_transition(o, t);
s = &context->states[12]; d = &context->states[15];
t = make_transition(&context->transitions[23], 400000025, 51, o, s, d); add_scope_transition(o, t);
s = &context->states[2]; d = &context->states[16];
t = make_transition(&context->transitions[24], 400000050, 52, o, s, d); add_scope_transition(o, t);
s = &context->states[6]; d = &context->states[16];
t = make_transition(&context->transitions[25], 400000046, 53, o, s, d); add_scope_transition(o, t);
s = &context->states[9]; d = &context->states[16];
t = make_transition(&context->transitions[26], 400000030, 54, o, s, d); add_scope_transition(o, t);
s = &context->states[13]; d = &context->states[16];
t = make_transition(&context->transitions[27], 400000026, 55, o, s, d); add_scope_transition(o, t);
s = &context->states[14]; d = &context->states[16];
t = make_transition(&context->transitions[28], 400000023, 56, o, s, d); add_scope_transition(o, t);
s = &context->states[7]; d = &context->states[16];
t = make_transition(&context->transitions[29], 400000022, 57, o, s, d); add_scope_transition(o, t);
}
/* create and link state expressions */
{
  sf_state * s;
  s = &context->states[0];
  s = &context->states[1];
  set_entry_action(s, C1_s_entry_1);
  set_during_action(s, C1_s_during_1);
  s = &context->states[2];

```

```

set_entry_action(s, C1_s_entry_2);

```

```

set_during_action(s, C1_s_during_2);
s = &context->states[3];
set_entry_action(s, C1_s_entry_3);
set_during_action(s, C1_s_during_3);
s = &context->states[4];
set_entry_action(s, C1_s_entry_4);
s = &context->states[5];
set_entry_action(s, C1_s_entry_5);
s = &context->states[6];
set_entry_action(s, C1_s_entry_6);
s = &context->states[7];
set_entry_action(s, C1_s_entry_7);
s = &context->states[8];
set_entry_action(s, C1_s_entry_8);
s = &context->states[9];
set_entry_action(s, C1_s_entry_9);
set_during_action(s, C1_s_during_9);
s = &context->states[10];
set_entry_action(s, C1_s_entry_10);
set_during_action(s, C1_s_during_10);
s = &context->states[11];
set_entry_action(s, C1_s_entry_11);
set_during_action(s, C1_s_during_11);
s = &context->states[12];
set_entry_action(s, C1_s_entry_12);
s = &context->states[13];
set_entry_action(s, C1_s_entry_13);

```

```

set_entry_action(s, C1_s_entry_14);
s = &context->states[15];
set_entry_action(s, C1_s_entry_15);
s = &context->states[16];
set_entry_action(s, C1_s_entry_16);
}
/* create and link transition expressions */
{
sf_transition * t;
t = &context->transitions[0];
set_guard(t, C1_t_guard_0);
t = &context->transitions[1];
set_guard(t, C1_t_guard_1);
t = &context->transitions[2];
set_guard(t, C1_t_guard_2);
t = &context->transitions[3];
set_guard(t, C1_t_guard_3);
t = &context->transitions[4];
set_guard(t, C1_t_guard_4);
t = &context->transitions[5];
set_guard(t, C1_t_guard_5);
t = &context->transitions[6];
set_guard(t, C1_t_guard_6);
t = &context->transitions[7];
set_guard(t, C1_t_guard_7);
t = &context->transitions[8];
set_guard(t, C1_t_guard_8);
t = &context->transitions[9];
set_guard(t, C1_t_guard_9);
t = &context->transitions[10];
set_guard(t, C1_t_guard_10);
t = &context->transitions[11];
set_guard(t, C1_t_guard_11);

```

```

t = &context->transitions[12];

```

```

set_guard(t, C1_t_guard_12);
t = &context->transitions[13];
set_guard(t, C1_t_guard_13);
t = &context->transitions[14];
set_guard(t, C1_t_guard_14);
t = &context->transitions[15];
set_guard(t, C1_t_guard_15);
t = &context->transitions[16];
set_guard(t, C1_t_guard_16);
t = &context->transitions[17];
set_guard(t, C1_t_guard_17);
t = &context->transitions[18];
set_guard(t, C1_t_guard_18);
t = &context->transitions[19];
set_guard(t, C1_t_guard_19);
t = &context->transitions[20];
set_guard(t, C1_t_guard_20);
t = &context->transitions[21];
set_guard(t, C1_t_guard_21);
t = &context->transitions[22];
set_guard(t, C1_t_guard_22);
t = &context->transitions[23];
set_guard(t, C1_t_guard_23);
t = &context->transitions[24];
set_guard(t, C1_t_guard_24);

```

```

set_guard(t, C1_t_guard_25);
t = &context->transitions[26];
set_guard(t, C1_t_guard_26);
t = &context->transitions[27];
set_guard(t, C1_t_guard_27);
t = &context->transitions[28];
set_guard(t, C1_t_guard_28);
t = &context->transitions[29];
set_guard(t, C1_t_guard_29);
}
}

bool C1_sf_input(sf_context *context)
{
    sf_data * d;
    sf_event * e;
    d = &context->data[9]; /* gear */
    d->data.bv = (char)context->inputs[0];
    d = &context->data[8]; /* V */
    d->data.dv = (double)context->inputs[1];
    d = &context->data[7]; /* shift_speed_12 */
    d->data.dv = (double)context->inputs[2];
    d = &context->data[6]; /* shift_speed_21 */
    d->data.dv = (double)context->inputs[3];
    d = &context->data[3]; /* shift_speed_23 */
    d->data.dv = (double)context->inputs[4];
    d = &context->data[2]; /* shift_speed_32 */
    d->data.dv = (double)context->inputs[5];
    d = &context->data[1]; /* shift_speed_34 */
    d->data.dv = (double)context->inputs[6];
    d = &context->data[0]; /* shift_speed_43 */
    d->data.dv = (double)context->inputs[7];
    return true;
}

```

```
bool C1_sf_output(sf_context *context)
```

```

{
    sf_data * d;
    sf_event * e;
    d = &context->data[10]; /* to_gear */
    context->outputs[1] = (double)d->data.bv;
    return true;
}

```

## 9.3. TargetLink Code

### 9.3.1. subsystem.h

```
/******\
```

Model code interface : Subsystem.h  
for TargetLink model : forges1b/Subsystem

Generated by TargetLink, the dSPACE production quality code generator  
Wed Jun 05 14:52:21 2002

CODE GENERATOR OPTIONS:

Target : Generic  
ANSI-C compatible code : yes  
Optimization level : 5  
Constant style : decimal  
Clean code option : disabled  
Logging mode : According to block-specific data  
Linker sections : enabled  
Interrupt functions : enabled  
User attributes : enabled  
Assembler statements : disabled  
Variable name length : 31 chars  
Separate lookup search function : disabled  
Use global bitfields : disabled  
Stateflow: use of bitfields : enabled  
State activity encoding limit : 5  
Omit zero inits in restart function: disabled  
Share fcns between TL subsystems : disabled  
Generate 64bit functions : enabled  
Inlining Threshold : 6  
Line break limit : 100  
Constant suffix : disabled  
Target optimized boolean data type : enabled

Subsys Corresponding Simulink Subsystem  
Sa1 Subsystem

SF-Node	Corresponding Stateflow Node	description
Ca0	Subsystem/Stateflow machine [forges1b]	
Ca1	Subsystem/shift_schedule	
Ca2	Subsystem/shift_schedule.first_gear	
Ca3	Subsystem/shift_schedule.fourth_gear	

Ca4 Subsystem/shift\_schedule.second\_gear

Ca5 Subsystem/shift\_schedule.shift\_pending1  
Ca6 Subsystem/shift\_schedule.shift\_pending2  
Ca7 Subsystem/shift\_schedule.shift\_pending3  
Ca8 Subsystem/shift\_schedule.shift\_pending4  
Ca9 Subsystem/shift\_schedule.shift\_pending5  
Ca10 Subsystem/shift\_schedule.shift\_pending6  
Ca11 Subsystem/shift\_schedule.shifting1  
Ca12 Subsystem/shift\_schedule.shifting2  
Ca13 Subsystem/shift\_schedule.shifting3  
Ca14 Subsystem/shift\_schedule.shifting4  
Ca15 Subsystem/shift\_schedule.shifting5  
Ca16 Subsystem/shift\_schedule.shifting6  
Ca17 Subsystem/shift\_schedule.third\_gear

TargetLink version : 1.2p1 from 19-Jul-2001  
Codegenerator version : 1.2.P1 from Jul 19 2001, 17:45:00  
Copyright (c) 1999-2001 by dSPACE GmbH

\\\*\*\*\*\*/

```

#define _SUBSYSTEM_H_

/*****\
 includes
 \*****/

#include "tl_types.h" /* definition of base data types */
#include "Subsystem_udt.h" /* type definition of the user data types */

/*****\
 type definitions
 \*****/
typedef struct{
 unsigned int Ca1_shift_schedule_ns : 5; /* LSB: 2^0 OFF: 0 MIN/MAX: 0 .. 31 */
 unsigned int Ca1_shift_schedule : 1; /* boolean type */
} BFa7_tp;

/*****\
 TL_CG_MACROCL_GLOBAL: Default macro class for macros with module extent.
 \*****/
#ifndef SaROOT_FX_GROUND
#define SaROOT_FX_GROUND 0
#endif /* SaROOT_FX_GROUND */

#define __no_state 0

/*****\
 defaultClass_sGlobal: Default storage class for global variables.
 \*****/
extern BFa7_tp BFa7; /* structure type */

/*****\
 exported functions
 \*****/
/* Model step function of subsystem(s): Subsystem */
extern void Subsystem(
 F64 Sa1_agear_ /* 64 bit floating point variable */,
 F64 Sa1_V_ /* 64 bit floating point variable */,

F64 Sa1_ss_12_ /* 64 bit floating point variable */,

F64 Sa1_ss_21_ /* 64 bit floating point variable */,
F64 Sa1_ss_23_ /* 64 bit floating point variable */,
F64 Sa1_ss_32_ /* 64 bit floating point variable */,
F64 Sa1_ss_34_ /* 64 bit floating point variable */,
F64 Sa1_ss_43_ /* 64 bit floating point variable */,
F64 * Sa1_ngear_ /* pointer to 64 bit floating point variable */);

#endif /* _SUBSYSTEM_H_ */

```

### 9.3.2. subsystem.c

```
/******\

Model code file   : Subsystem.c
for TargetLink model : forges1b/Subsystem

Generated by TargetLink, the dSPACE production quality code generator
Wed Jun 05 14:52:21 2002

CODE GENERATOR OPTIONS:
Target           : Generic
ANSI-C compatible code : yes
Optimization level   : 5
Constant style      : decimal
Clean code option    : disabled
Logging mode        : According to block-specific data
Linker sections     : enabled
Interrupt functions  : enabled
User attributes     : enabled
Assembler statements : disabled
Variable name length : 31 chars
Separate lookup search function : disabled
Use global bitfields : disabled
Stateflow: use of bitfields : enabled
State activity encoding limit : 5
Omit zero inits in restart function: disabled
Share fcns between TL subsystems : disabled
Generate 64bit functions : enabled
Inlining Threshold   : 6
Line break limit     : 100
Constant suffix      : disabled
Target optimized boolean data type : enabled

Subsys Corresponding Simulink Subsystem
Sa1 Subsystem

SF-Node Corresponding Stateflow Node      description
Ca0 Subsystem/Stateflow machine [forges1b]
Ca1 Subsystem/shift_schedule
Ca2 Subsystem/shift_schedule.first_gear
Ca3 Subsystem/shift_schedule.fourth_gear
Ca4 Subsystem/shift_schedule.second_gear

Ca5 Subsystem/shift_schedule.shift_pending1

Ca6 Subsystem/shift_schedule.shift_pending2
Ca7 Subsystem/shift_schedule.shift_pending3
Ca8 Subsystem/shift_schedule.shift_pending4
Ca9 Subsystem/shift_schedule.shift_pending5
Ca10 Subsystem/shift_schedule.shift_pending6
Ca11 Subsystem/shift_schedule.shifting1
Ca12 Subsystem/shift_schedule.shifting2
Ca13 Subsystem/shift_schedule.shifting3
Ca14 Subsystem/shift_schedule.shifting4
Ca15 Subsystem/shift_schedule.shifting5
Ca16 Subsystem/shift_schedule.shifting6
Ca17 Subsystem/shift_schedule.third_gear

TargetLink version : 1.2p1 from 19-Jul-2001
```

```

Copyright (c) 1999-2001 by dSPACE GmbH
/*****

#define _SUBSYSTEM_C_      /* identifier for this file */

/*****\
includes
/*****\

#include "tl_types.h"      /* definition of base data types */

#ifdef TL_FRAME
#include "Subsystem_frm.h" /* definitions for the simulation frame */
#endif

#include "Subsystem.h" /* Include of own header file */
#include "Subsystem_udt.h" /* type definition of the user data types */

/*****\
TL_CG_MACROCL_STATIC: Default macro class for macros with module extent.
/*****\
#define Ca1_DELAY      1
#define Ca3_fourth_gear_id      12
#define Ca6_shift_pending2_id      7
#define Ca13_shifting3_id      11
#define Ca5_shift_pending1_id      6
#define Ca17_third_gear_id      1
#define Ca8_shift_pending4_id      16
#define Ca10_shift_pending6_id      14
#define Ca11_shifting1_id      9
#define Ca9_shift_pending5_id      15
#define Ca4_second_gear_id      13
#define Ca15_shifting5_id      4
#define Ca16_shifting6_id      3
#define Ca7_shift_pending3_id      8
#define Ca2_first_gear_id      2
#define Ca14_shifting4_id      5
#define Ca12_shifting2_id      10

/*****\
defaultClass_sfStaticGlobalInit: Default storage class for global static variables with initial
value
/*****\
static S8      Ca1_ctr; /* LSB: 2^0 OFF: 0 MIN/MAX: -128 .. 127 */

/*****\

defaultClass_sfStaticGlobal: Default storage class for global static variables
/*****\
static S8      Ca1_to_gear; /* LSB: 2^0 OFF: 0 MIN/MAX: -128 .. 127 */

/*****\
defaultClass_slGlobal: Default storage class for global variables.
/*****\

void Subsystem_init()
{
    BFa7.Ca1_shift_schedule_ns = 0;
    BFa7.Ca1_shift_schedule = 0;

```

```

Ca1_to_gear = 1;

};

/*****
Model step function of subsystem(s): Subsystem
*****/
void Subsystem(
F64 Sa1_agear_ /* 64 bit floating point variable */,
F64 Sa1_V_ /* 64 bit floating point variable */,
F64 Sa1_ss_12_ /* 64 bit floating point variable */,
F64 Sa1_ss_21_ /* 64 bit floating point variable */,
F64 Sa1_ss_23_ /* 64 bit floating point variable */,
F64 Sa1_ss_32_ /* 64 bit floating point variable */,
F64 Sa1_ss_34_ /* 64 bit floating point variable */,
F64 Sa1_ss_43_ /* 64 bit floating point variable */,
F64 * Sa1_ngear_ /* pointer to 64 bit floating point variable */)
{
/* Stateflow: Subsystem/shift_schedule */
/* Begin execution of chart Subsystem/shift_schedule */
switch ( BFa7.Ca1_shift_schedule_ns ) {
case Ca2_first_gear_id: {
/* Begin execution of state Subsystem/shift_schedule.first_gear */
if ( Sa1_V_ > Sa1_ss_12_ ) {
/*
State transition from Subsystem/shift_schedule.first_gear to Subsystem/shift_schedule
.shift_pending1 */
BFa7.Ca1_shift_schedule_ns = Ca5_shift_pending1_id;
Ca1_ctr = 0;
Ca1_to_gear = 1;
}
/* End execution of state Subsystem/shift_schedule.first_gear */
break;
}

case Ca3_fourth_gear_id: {
/* Begin execution of state Subsystem/shift_schedule.fourth_gear */
if ( Sa1_V_ <= Sa1_ss_43_ ) {
/*
State transition from Subsystem/shift_schedule.fourth_gear to Subsystem/shift_schedul
e.shift_pending6 */
BFa7.Ca1_shift_schedule_ns = Ca10_shift_pending6_id;
Ca1_ctr = 0;
Ca1_to_gear = 4;
}
/* End execution of state Subsystem/shift_schedule.fourth_gear */
break;
}

}
}

```

```

case Ca4_second_gear_id: {
/* Begin execution of state Subsystem/shift_schedule.second_gear */
if ( Sa1_V_ > Sa1_ss_23_ ) {
/*
State transition from Subsystem/shift_schedule.second_gear to Subsystem/shift_schedul
e.shift_pending2 */
BFa7.Ca1_shift_schedule_ns = Ca6_shift_pending2_id;
Ca1_ctr = 0;
Ca1_to_gear = 2;
}
}

```

```

if ( Sa1_V_ <= Sa1_ss_21_ ) {
    /*
     State transition from Subsystem/shift_schedule.second_gear to Subsystem/shift_sche
     dule.shift_pending4 */
    BFa7.Ca1_shift_schedule_ns = Ca8_shift_pending4_id;
    Ca1_ctr = 0;
    Ca1_to_gear = 2;
}
}
/* End execution of state Subsystem/shift_schedule.second_gear */
break;
}

case Ca5_shift_pending1_id: {
    /* Begin execution of state Subsystem/shift_schedule.shift_pending1 */
    if ( Ca1_ctr > Ca1_DELAY ) {
        /*
         State transition from Subsystem/shift_schedule.shift_pending1 to Subsystem/shift_sche
         dule.shifting1 */
        BFa7.Ca1_shift_schedule_ns = Ca11_shifting1_id;
        Ca1_to_gear = 2;
    } else {
        if ( Sa1_V_ <= Sa1_ss_21_ ) {
            /*
             State transition from Subsystem/shift_schedule.shift_pending1 to Subsystem/shift_s
             chedule.first_gear */
            BFa7.Ca1_shift_schedule_ns = Ca2_first_gear_id;
            Ca1_to_gear = 1;
        } else {
            Ca1_ctr = Ca1_ctr + 1;
        }
    }
}
/* End execution of state Subsystem/shift_schedule.shift_pending1 */
break;
}

case Ca6_shift_pending2_id: {
    /* Begin execution of state Subsystem/shift_schedule.shift_pending2 */
    if ( Ca1_ctr > Ca1_DELAY ) {
        /*
         State transition from Subsystem/shift_schedule.shift_pending2 to Subsystem/shift_sche
         dule.shifting2 */
        BFa7.Ca1_shift_schedule_ns = Ca12_shifting2_id;
        Ca1_to_gear = 3;
    } else {
        if ( Sa1_V_ <= Sa1_ss_23_ ) {
            /*
             State transition from Subsystem/shift_schedule.shift_pending2 to Subsystem/shift_s
             chedule.second_gear */
            BFa7.Ca1_shift_schedule_ns = Ca4_second_gear_id;

Ca1_to_gear = 2;

        } else {
            Ca1_ctr = Ca1_ctr + 1;
        }
    }
}
/* End execution of state Subsystem/shift_schedule.shift_pending2 */
break;
}

```

```

/* Begin execution of state Subsystem/shift_schedule.shift_pending3 */
if ( Ca1_ctr > Ca1_DELAY ) {
  /*
   State transition from Subsystem/shift_schedule.shift_pending3 to Subsystem/shift_sche
   dule.shifting3 */
  BFa7.Ca1_shift_schedule_ns = Ca13_shifting3_id;
  Ca1_to_gear = 4;
} else {
  if ( Sa1_V_ <= Sa1_ss_43_ ) {
    /*
     State transition from Subsystem/shift_schedule.shift_pending3 to Subsystem/shift_s
     chedule.third_gear */
    BFa7.Ca1_shift_schedule_ns = Ca17_third_gear_id;
    Ca1_to_gear = 3;
  } else {
    Ca1_ctr = Ca1_ctr + 1;
  }
}
/* End execution of state Subsystem/shift_schedule.shift_pending3 */
break;
}

```

```

case Ca8_shift_pending4_id: {
  /* Begin execution of state Subsystem/shift_schedule.shift_pending4 */
  if ( Sa1_V_ > Sa1_ss_12_ ) {
    /*
     State transition from Subsystem/shift_schedule.shift_pending4 to Subsystem/shift_sche
     dule.second_gear */
    BFa7.Ca1_shift_schedule_ns = Ca4_second_gear_id;
    Ca1_to_gear = 2;
  } else {
    if ( Ca1_ctr > Ca1_DELAY ) {
      /*
       State transition from Subsystem/shift_schedule.shift_pending4 to Subsystem/shift_s
       chedule.shifting4 */
      BFa7.Ca1_shift_schedule_ns = Ca14_shifting4_id;
      Ca1_to_gear = 1;
    } else {
      Ca1_ctr = Ca1_ctr + 1;
    }
  }
}
/* End execution of state Subsystem/shift_schedule.shift_pending4 */
break;
}

```

```

case Ca9_shift_pending5_id: {
  /* Begin execution of state Subsystem/shift_schedule.shift_pending5 */
  if ( Sa1_V_ > Sa1_ss_23_ ) {
    /*
     State transition from Subsystem/shift_schedule.shift_pending5 to Subsystem/shift_sche
     dule.third_gear */
    BFa7.Ca1_shift_schedule_ns = Ca17_third_gear_id;

```

```

Ca1_to_gear = 3;

```

```

} else {
  if ( Ca1_ctr > Ca1_DELAY ) {
    /*
     State transition from Subsystem/shift_schedule.shift_pending5 to Subsystem/shift_s
     chedule.shifting5 */
    BFa7.Ca1_shift_schedule_ns = Ca15_shifting5_id;

```

```

    } else {
        Ca1_ctr = Ca1_ctr + 1;
    }
}
/* End execution of state Subsystem/shift_schedule.shift_pending5 */
break;
}

case Ca10_shift_pending6_id: {
    /* Begin execution of state Subsystem/shift_schedule.shift_pending6 */
    if ( Sa1_V_ > Sa1_ss_34_ ) {
        /*
         * State transition from Subsystem/shift_schedule.shift_pending6 to Subsystem/shift_sche
         * dule.fourth_gear */
        BFa7.Ca1_shift_schedule_ns = Ca3_fourth_gear_id;
        Ca1_to_gear = 4;
    } else {
        if ( Ca1_ctr > Ca1_DELAY ) {
            /*
             * State transition from Subsystem/shift_schedule.shift_pending6 to Subsystem/shift_s
             * chedule.shifting6 */
            BFa7.Ca1_shift_schedule_ns = Ca16_shifting6_id;
            Ca1_to_gear = 3;
        } else {
            Ca1_ctr = Ca1_ctr + 1;
        }
    }
}
/* End execution of state Subsystem/shift_schedule.shift_pending6 */
break;
}

case Ca11_shifting1_id: {
    /* Begin execution of state Subsystem/shift_schedule.shifting1 */
    if ( Sa1_agenar_ == 2. ) {
        /*
         * State transition from Subsystem/shift_schedule.shifting1 to Subsystem/shift_sche
         * dule.second_gear */
        BFa7.Ca1_shift_schedule_ns = Ca4_second_gear_id;
        Ca1_to_gear = 2;
    } else {
        if ( Sa1_V_ <= Sa1_ss_21_ ) {
            /*
             * State transition from Subsystem/shift_schedule.shifting1 to Subsystem/shift_sche
             * dule.first_gear */
            BFa7.Ca1_shift_schedule_ns = Ca2_first_gear_id;
            Ca1_to_gear = 1;
        }
    }
}
/* End execution of state Subsystem/shift_schedule.shifting1 */
break;
}

case Ca12_shifting2_id: {
    /* Begin execution of state Subsystem/shift_schedule.shifting2 */

```

```

if ( Sa1_agenar_ == 3. ) {

```

```

    /*
     * State transition from Subsystem/shift_schedule.shifting2 to Subsystem/shift_sche
     * dule.third_gear */
    BFa7.Ca1_shift_schedule_ns = Ca17_third_gear_id;

```

```

} else {
  if ( Sa1_V_ <= Sa1_ss_23_ ) {
    /*
     State transition from Subsystem/shift_schedule.shifting2 to Subsystem/shift_schedu
     le.second_gear */
    BFa7.Ca1_shift_schedule_ns = Ca4_second_gear_id;
    Ca1_to_gear = 2;
  }
}
/* End execution of state Subsystem/shift_schedule.shifting2 */
break;
}

case Ca13_shifting3_id: {
  /* Begin execution of state Subsystem/shift_schedule.shifting3 */
  if ( Sa1_agear_ == 4. ) {
    /*
     State transition from Subsystem/shift_schedule.shifting3 to Subsystem/shift_schedule.
     fourth_gear */
    BFa7.Ca1_shift_schedule_ns = Ca3_fourth_gear_id;
    Ca1_to_gear = 4;
  } else {
    if ( Sa1_V_ <= Sa1_ss_43_ ) {
      /*
       State transition from Subsystem/shift_schedule.shifting3 to Subsystem/shift_schedu
       le.third_gear */
      BFa7.Ca1_shift_schedule_ns = Ca17_third_gear_id;
      Ca1_to_gear = 3;
    }
  }
  /* End execution of state Subsystem/shift_schedule.shifting3 */
  break;
}

case Ca14_shifting4_id: {
  /* Begin execution of state Subsystem/shift_schedule.shifting4 */
  if ( Sa1_V_ > Sa1_ss_12_ ) {
    /*
     State transition from Subsystem/shift_schedule.shifting4 to Subsystem/shift_schedule.
     second_gear */
    BFa7.Ca1_shift_schedule_ns = Ca4_second_gear_id;
    Ca1_to_gear = 2;
  } else {
    if ( Sa1_agear_ == 1. ) {
      /*
       State transition from Subsystem/shift_schedule.shifting4 to Subsystem/shift_schedu
       le.first_gear */
      BFa7.Ca1_shift_schedule_ns = Ca2_first_gear_id;
      Ca1_to_gear = 1;
    }
  }
  /* End execution of state Subsystem/shift_schedule.shifting4 */
  break;
}

case Ca15_shifting5_id: {

```

```

/* Begin execution of state Subsystem/shift_schedule.shifting5 */

```

```

if ( Sa1_V_ > Sa1_ss_23_ ) {
  /*

```

```

        third_gear */
        BFa7.Ca1_shift_schedule_ns = Ca17_third_gear_id;
        Ca1_to_gear = 3;
    } else {
        if ( Sa1_agear_ == 2. ){
            /*
             * State transition from Subsystem/shift_schedule.shifting5 to Subsystem/shift_sche
             * le.second_gear */
            BFa7.Ca1_shift_schedule_ns = Ca4_second_gear_id;
            Ca1_to_gear = 2;
        }
    }
    /* End execution of state Subsystem/shift_schedule.shifting5 */
    break;
}

case Ca16_shifting6_id: {
    /* Begin execution of state Subsystem/shift_schedule.shifting6 */
    if ( Sa1_V_ > Sa1_ss_34_ ){
        /*
         * State transition from Subsystem/shift_schedule.shifting6 to Subsystem/shift_sche
         * dule.fourth_gear */
        BFa7.Ca1_shift_schedule_ns = Ca3_fourth_gear_id;
        Ca1_to_gear = 4;
    } else {
        if ( Sa1_agear_ == 3. ){
            /*
             * State transition from Subsystem/shift_schedule.shifting6 to Subsystem/shift_sche
             * dule.third_gear */
            BFa7.Ca1_shift_schedule_ns = Ca17_third_gear_id;
            Ca1_to_gear = 3;
        }
    }
    /* End execution of state Subsystem/shift_schedule.shifting6 */
    break;
}

case Ca17_third_gear_id: {
    /* Begin execution of state Subsystem/shift_schedule.third_gear */
    if ( Sa1_V_ > Sa1_ss_34_ ){
        /*
         * State transition from Subsystem/shift_schedule.third_gear to Subsystem/shift_sche
         * dule.shift_pending3 */
        BFa7.Ca1_shift_schedule_ns = Ca7_shift_pending3_id;
        Ca1_ctr = 0;
        Ca1_to_gear = 3;
    } else {
        if ( Sa1_V_ <= Sa1_ss_32_ ){
            /*
             * State transition from Subsystem/shift_schedule.third_gear to Subsystem/shift_sche
             * dule.shift_pending5 */
            BFa7.Ca1_shift_schedule_ns = Ca9_shift_pending5_id;
            Ca1_ctr = 0;
            Ca1_to_gear = 3;
        }
    }
    /* End execution of state Subsystem/shift_schedule.third_gear */
    break;
}
}
}

```

```

default: {
  if ( !(BFa7.Ca1_shift_schedule) ){
    BFa7.Ca1_shift_schedule = 1;
    BFa7.Ca1_shift_schedule_ns = Ca2_first_gear_id;
    Ca1_to_gear = 1;
  }
}

}

/* End execution of chart Subsystem/shift_schedule */

/* TargetLink output: Subsystem/ngear_ */
*Sa1_ngear_ = (F64)Ca1_to_gear;
}

#endif _SUBSYSTEM_C_

```

## 9.4. RTW-Embedded Coder code

We have only included one of the nine files generated. If the reader is interested they may contact the author for copies. This particular file is the most significant in terms of the code behavior.

### 9.4.1. forges1a.c

```

/*
 * forges1a.c
 *
 * Real-Time Workshop code generation for Simulink model "forges1a.mdl".
 *
 * Model Version          : 1.6
 * Real-Time Workshop file version   : 4.0 $Date: 2000/09/19 19:45:27 $
 * Real-Time Workshop file generated on : Thu May 30 12:13:04 2002
 * TLC version            : 4.0 (Aug 21 2000)
 * C source code generated on      : Thu May 30 12:13:05 2002
 *
 * Relevant TLC Options:
 * InlineParameters      = 0
 * RollThreshold         = 5
 * CodeFormat            = Embedded-C
 *
 * Simulink model settings:
 * Solver                : FixedStep
 * StartTime             : 0.0 s
 * StopTime              : 10.0 s
 * FixedStep              : 0.2 s
 */

#include "forges1a.h"
#include "forges1a_prm.h"

```

```

/* Start of Functions in model "forges1a" */

/* model step function */
void forges1a_step(void)
{

/* update absolute time */
ssUpdateRealAbsoluteTime(forges1a_rt0);

/* Stateflow Chart Code: <Root>/shift_schedule */
{
SFshift_scheduleInstanceStruct      *chartInstance         =      (SFshift_scheduleInstanceStruct
*forges1a_DWork.s1_SFunction_PWORK.ChartInstance;

#define IN_NO_ACTIVE_CHILD          (0)
#define IN_c10_s1_first_gear        1
#define IN_c10_s2_fourth_gear       2
#define IN_c10_s3_second_gear       3
#define IN_c10_s4_shift_pending1    4
#define IN_c10_s5_shift_pending2    5
#define IN_c10_s6_shift_pending3    6
#define IN_c10_s7_shift_pending4    7
#define IN_c10_s8_shift_pending5    8
#define IN_c10_s9_shift_pending6    9
#define IN_c10_s10_shifting1        10
#define IN_c10_s11_shifting2        11
#define IN_c10_s12_shifting3        12
#define IN_c10_s13_shifting4        13
#define IN_c10_s14_shifting5        14
#define IN_c10_s15_shifting6        15
#define IN_c10_s16_third_gear       16
#define c10_d11_DELAY                3

{
int previousEvent;
previousEvent = _sfEvent_forges1a_;
/* Call this chart */
_sfEvent_forges1a_ = CALL_EVENT;
{
if(chartInstance->State.is_active_shift_schedule==0) {
chartInstance->State.is_active_shift_schedule=1;
chartInstance->State.is_shift_schedule=IN_c10_s1_first_gear;
forges1a_B.s1_SFunction_o2 = (real_T)1;
} else {
switch(chartInstance->State.is_shift_schedule) {
case IN_NO_ACTIVE_CHILD:
break;
case IN_c10_s1_first_gear:
if(forges1a_U.root_V > forges1a_U.root_ss_12) {
chartInstance->State.is_shift_schedule=IN_c10_s4_shift_pending1;
chartInstance->LocalData.c10_d1_ctr = 0;
forges1a_B.s1_SFunction_o2 = (real_T)1;
}
break;
case IN_c10_s2_fourth_gear:
if(forges1a_U.root_V <= forges1a_U.root_ss_43) {
chartInstance->State.is_shift_schedule=IN_c10_s9_shift_pending6;
chartInstance->LocalData.c10_d1_ctr = 0;
forges1a_B.s1_SFunction_o2 = (real_T)4;
}
break;

```

```

case IN_c10_s3_second_gear:
if(forges1a_U.root_V > forges1a_U.root_ss_23) {

```

```

    chartInstance->State.is_shift_schedule=IN_c10_s5_shift_pending2;
    chartInstance->LocalData.c10_d1_ctr = 0;
    forges1a_B.s1_SFunction_o2 = (real_T)2;
} else if(forges1a_U.root_V <= forges1a_U.root_ss_21) {
    chartInstance->State.is_shift_schedule=IN_c10_s7_shift_pending4;
    chartInstance->LocalData.c10_d1_ctr = 0;
    forges1a_B.s1_SFunction_o2 = (real_T)2;
}
break;
case IN_c10_s4_shift_pending1:
if(chartInstance->LocalData.c10_d1_ctr > c10_d11_DELAY) {
    chartInstance->State.is_shift_schedule=IN_c10_s10_shifting1;
    forges1a_B.s1_SFunction_o2 = (real_T)2;
} else if(forges1a_U.root_V <= forges1a_U.root_ss_21) {
    chartInstance->State.is_shift_schedule=IN_c10_s1_first_gear;
    forges1a_B.s1_SFunction_o2 = (real_T)1;
} else {
    chartInstance->LocalData.c10_d1_ctr = (int8_T)(chartInstance->LocalData.c10_d1_ctr + 1);
}
break;
case IN_c10_s5_shift_pending2:
if(chartInstance->LocalData.c10_d1_ctr > c10_d11_DELAY) {
    chartInstance->State.is_shift_schedule=IN_c10_s11_shifting2;
    forges1a_B.s1_SFunction_o2 = (real_T)3;
} else if(forges1a_U.root_V <= forges1a_U.root_ss_23) {
    chartInstance->State.is_shift_schedule=IN_c10_s3_second_gear;
    forges1a_B.s1_SFunction_o2 = (real_T)2;
} else {
    chartInstance->LocalData.c10_d1_ctr = (int8_T)(chartInstance->LocalData.c10_d1_ctr + 1);
}
break;
case IN_c10_s6_shift_pending3:
if(chartInstance->LocalData.c10_d1_ctr > c10_d11_DELAY) {
    chartInstance->State.is_shift_schedule=IN_c10_s12_shifting3;
    forges1a_B.s1_SFunction_o2 = (real_T)4;
} else if(forges1a_U.root_V <= forges1a_U.root_ss_43) {
    chartInstance->State.is_shift_schedule=IN_c10_s16_third_gear;
    forges1a_B.s1_SFunction_o2 = (real_T)3;
} else {
    chartInstance->LocalData.c10_d1_ctr = (int8_T)(chartInstance->LocalData.c10_d1_ctr + 1);
}
break;
case IN_c10_s7_shift_pending4:
if(forges1a_U.root_V > forges1a_U.root_ss_12) {
    chartInstance->State.is_shift_schedule=IN_c10_s3_second_gear;
    forges1a_B.s1_SFunction_o2 = (real_T)2;
} else if(chartInstance->LocalData.c10_d1_ctr > c10_d11_DELAY) {
    chartInstance->State.is_shift_schedule=IN_c10_s13_shifting4;
    forges1a_B.s1_SFunction_o2 = (real_T)1;
} else {
    chartInstance->LocalData.c10_d1_ctr = (int8_T)(chartInstance->LocalData.c10_d1_ctr + 1);
}
break;
case IN_c10_s8_shift_pending5:
if(forges1a_U.root_V > forges1a_U.root_ss_23) {
    chartInstance->State.is_shift_schedule=IN_c10_s16_third_gear;
    forges1a_B.s1_SFunction_o2 = (real_T)3;
} else if(chartInstance->LocalData.c10_d1_ctr > c10_d11_DELAY) {

```

```
forges1a_B.s1_SFunction_o2 = (real_T)2;
} else {
```

```
chartInstance->LocalData.c10_d1_ctr = (int8_T)(chartInstance->LocalData.c10_d1_ctr + 1);
```

```
}
break;
case IN_c10_s9_shift_pending6:
if(forges1a_U.root_V > forges1a_U.root_ss_34) {
chartInstance->State.is_shift_schedule=IN_c10_s2_fourth_gear;
forges1a_B.s1_SFunction_o2 = (real_T)4;
} else if(chartInstance->LocalData.c10_d1_ctr > c10_d11_DELAY) {
chartInstance->State.is_shift_schedule=IN_c10_s15_shifting6;
forges1a_B.s1_SFunction_o2 = (real_T)3;
} else {
chartInstance->LocalData.c10_d1_ctr = (int8_T)(chartInstance->LocalData.c10_d1_ctr + 1);
}
break;
case IN_c10_s10_shifting1:
if((int8_T)forges1a_U.root_agear == 2) {
chartInstance->State.is_shift_schedule=IN_c10_s3_second_gear;
forges1a_B.s1_SFunction_o2 = (real_T)2;
} else if(forges1a_U.root_V <= forges1a_U.root_ss_21) {
chartInstance->State.is_shift_schedule=IN_c10_s1_first_gear;
forges1a_B.s1_SFunction_o2 = (real_T)1;
}
break;
case IN_c10_s11_shifting2:
if((int8_T)forges1a_U.root_agear == 3) {
chartInstance->State.is_shift_schedule=IN_c10_s16_third_gear;
forges1a_B.s1_SFunction_o2 = (real_T)3;
} else if(forges1a_U.root_V <= forges1a_U.root_ss_23) {
chartInstance->State.is_shift_schedule=IN_c10_s3_second_gear;
forges1a_B.s1_SFunction_o2 = (real_T)2;
}
break;
case IN_c10_s12_shifting3:
if((int8_T)forges1a_U.root_agear == 4) {
chartInstance->State.is_shift_schedule=IN_c10_s2_fourth_gear;
forges1a_B.s1_SFunction_o2 = (real_T)4;
} else if(forges1a_U.root_V <= forges1a_U.root_ss_43) {
chartInstance->State.is_shift_schedule=IN_c10_s16_third_gear;
forges1a_B.s1_SFunction_o2 = (real_T)3;
}
break;
case IN_c10_s13_shifting4:
if(forges1a_U.root_V > forges1a_U.root_ss_12) {
chartInstance->State.is_shift_schedule=IN_c10_s3_second_gear;
forges1a_B.s1_SFunction_o2 = (real_T)2;
} else if((int8_T)forges1a_U.root_agear == 1) {
chartInstance->State.is_shift_schedule=IN_c10_s1_first_gear;
forges1a_B.s1_SFunction_o2 = (real_T)1;
}
break;
case IN_c10_s14_shifting5:
if(forges1a_U.root_V > forges1a_U.root_ss_23) {
chartInstance->State.is_shift_schedule=IN_c10_s16_third_gear;
forges1a_B.s1_SFunction_o2 = (real_T)3;
} else if((int8_T)forges1a_U.root_agear == 2) {
chartInstance->State.is_shift_schedule=IN_c10_s3_second_gear;
forges1a_B.s1_SFunction_o2 = (real_T)2;
```

```

break;
case IN_c10_s15_shifting6:
if(forges1a_U.root_V > forges1a_U.root_ss_34) {
chartInstance->State.is_shift_schedule=IN_c10_s2_fourth_gear;

```

```

forges1a_B.s1_SFunction_o2 = (real_T)4;

```

```

} else if((int8_T)forges1a_U.root_agear == 3) {
chartInstance->State.is_shift_schedule=IN_c10_s16_third_gear;
forges1a_B.s1_SFunction_o2 = (real_T)3;
}
break;
case IN_c10_s16_third_gear:
if(forges1a_U.root_V > forges1a_U.root_ss_34) {
chartInstance->State.is_shift_schedule=IN_c10_s6_shift_pending3;
chartInstance->LocalData.c10_d1_ctr = 0;
forges1a_B.s1_SFunction_o2 = (real_T)3;
} else if(forges1a_U.root_V <= forges1a_U.root_ss_32) {
chartInstance->State.is_shift_schedule=IN_c10_s8_shift_pending5;
chartInstance->LocalData.c10_d1_ctr = 0;
forges1a_B.s1_SFunction_o2 = (real_T)3;
}
break;
}
}
}

_sfEvent_forges1a_ = previousEvent;
}

#undef IN_c10_s1_first_gear
#undef IN_c10_s2_fourth_gear
#undef IN_c10_s3_second_gear
#undef IN_c10_s4_shift_pending1
#undef IN_c10_s5_shift_pending2
#undef IN_c10_s6_shift_pending3
#undef IN_c10_s7_shift_pending4
#undef IN_c10_s8_shift_pending5
#undef IN_c10_s9_shift_pending6
#undef IN_c10_s10_shifting1
#undef IN_c10_s11_shifting2
#undef IN_c10_s12_shifting3
#undef IN_c10_s13_shifting4
#undef IN_c10_s14_shifting5
#undef IN_c10_s15_shifting6
#undef IN_c10_s16_third_gear
#undef c10_d11_DELAY
}
/* Output Block: <Root>/ngear */
forges1a_Y.root_ngear = forges1a_B.s1_SFunction_o2;

/* logging */

/* Matfile logging */
rt_UpdateTXYLogVars(forges1a_rtO.L);

/* (no update code required) */

/* signal main to stop simulation */
if (!(ssGetTFinal(forges1a_rtO)-ssGetT(forges1a_rtO) > ssGetT(forges1a_rtO)*DBL_EPSILON)) {
ssSetErrorStatus(forges1a_rtO, "Simulation finished");
}

```

```
}  
/* End of Functions in model "forges1a" */  
#include "forges1a_reg.h"
```

```
/* [EOF] forges1a.c */
```

## 10. References

[1] [Challenge Problem Report](#)

[2] [http://vehicle.me.berkeley.edu/mobies/powertrain/powertrain\\_v\\_2\\_0.zip](http://vehicle.me.berkeley.edu/mobies/powertrain/powertrain_v_2_0.zip)