

Latest Features in Real-Time Workshop

October 2008

R2008b

Code Generation for Enumerated Data Types

Challenge

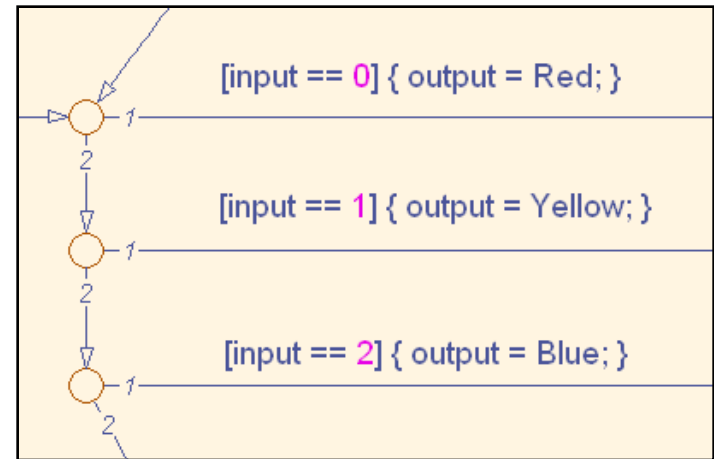
- End users need to clearly define data with a limited set of allowable values to improve modeling of modes and logic.

Solution

- Add support for enumerated data types in Simulink, Stateflow, and Real-Time Workshop

Benefit

- Improves clarity, especially for state, mode, and logic-centric models
- Improves robustness and readability
- Enables `enum` types to appear in code



Type definition in code

```
typedef enum {
    BasicColors_Red = 0,
    BasicColors_Yellow = 1,
    BasicColors_Blue = 2,
    BasicColors_BONUS = 10,
    BasicColors_ERROR = -1
} BasicColors;
```

Fixed-Point Code Generation for Word Sizes up to 128 Bits

Challenge

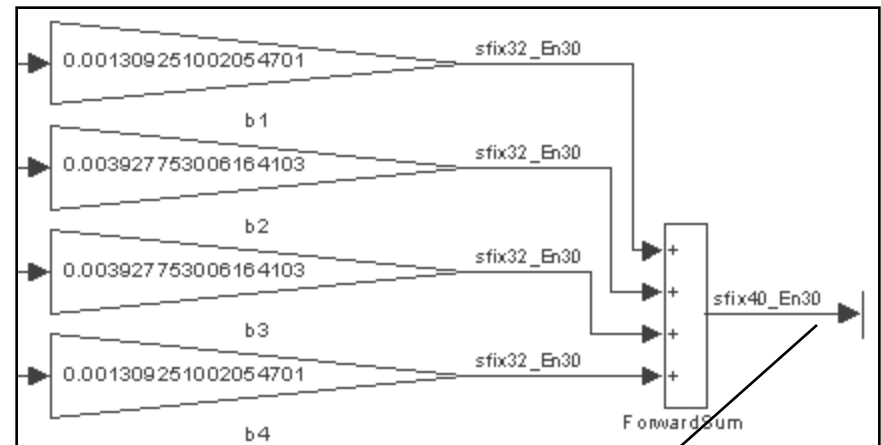
- Code cannot be generated beyond 32 bits for accelerated simulation modes or embedded deployment.
- Simulink simulates up to 128 bits.

Solution

- Generate code for all word lengths that simulation supports
- Use multiword when necessary

Benefit

- Accelerates simulations for more applications
- Leverages large word sizes on DSP chips and microprocessors, including 40+ bit accumulators
- Expands production code usage



long:

```
int16_T rtb_DownCastToMem;
rtb_DownCastToMem = (int16_T) (((int40_T) (5108 * DWork.UD3_DSTATE) +
2097152L) + (int40_T) (-17726 * DWork.UD2_DSTATE)) + (int40_T) (20724 *
DWork.UD1_DSTATE) >> 13);
```


Optimized Code for Vector Assignments Using `memcpy`

Challenge

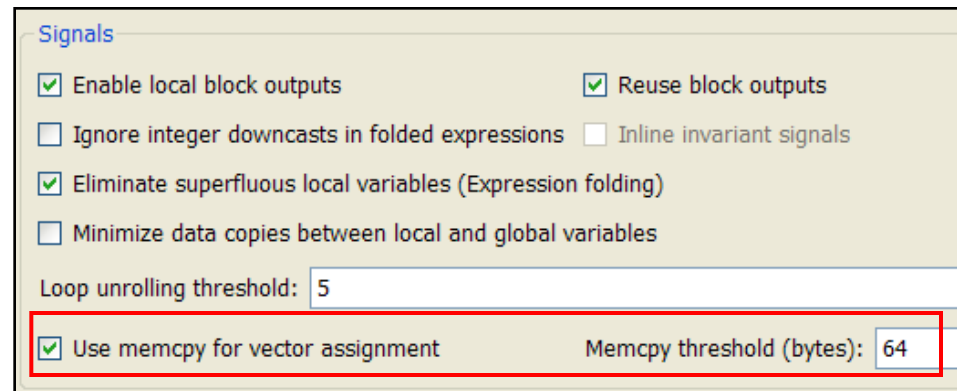
- `memcpy()` is often more efficient than for-loop controlled element assignment for large data sets.

Solution

- Provide option for emitting `memcpy()` when assigning a vector signal

Benefit

- Leverages `memcpy()` to improve execution speed
- Permits target-specific `memcpy()` through Target Function Library replacement



Example result for 1000 byte vector on Analog Devices BF53x

for-loop (cycles)	2026
<code>memcpy</code> (cycles)	309
Speed improvement	556%

Register Keywords to Avoid Conflicts

Challenge

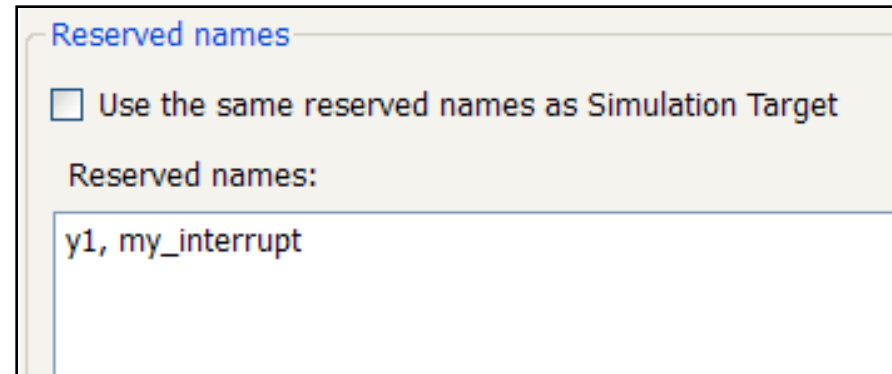
- External environment may prevent Real-Time Workshop from using certain keywords in generated code.

Solution

- Allow end users to specify the keywords that Real-Time Workshop should not use (e.g., y1)

Benefit

- Facilitates code integration when external functions and global variables are involved



Enhanced Rate Transition Asynchronous Branching

Challenge

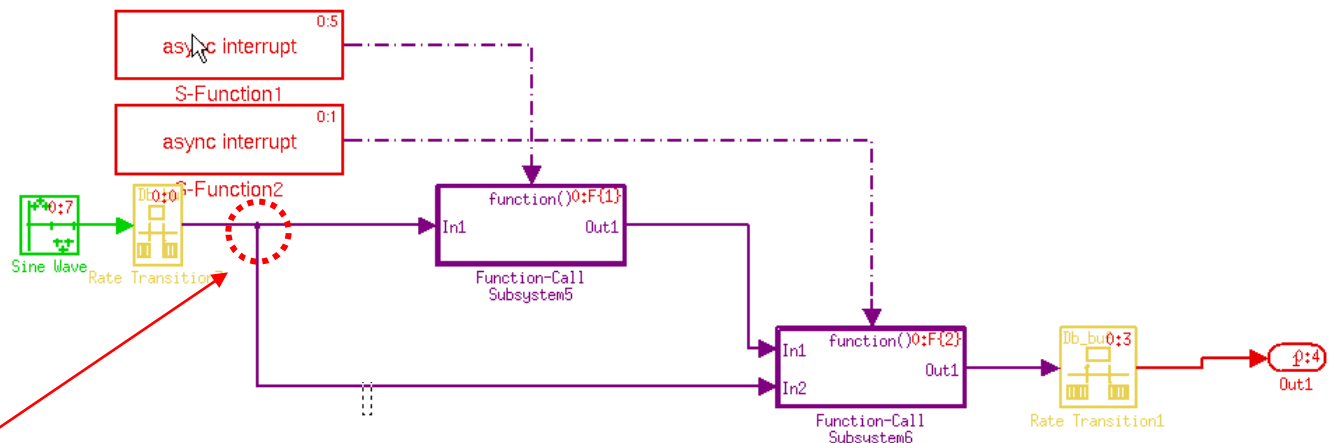
- If a signal drives multiple asynchronous subsystems, multiple rate transition blocks were required.

Solution

- Support branching of asynchronous Rate Transition blocks

Benefit

- Simplifies diagrams
- Improves code efficiency



Each branch no longer requires a separate Rate Transition block

Enhanced Rate Transition Asynchronous Direct Connection

Challenge

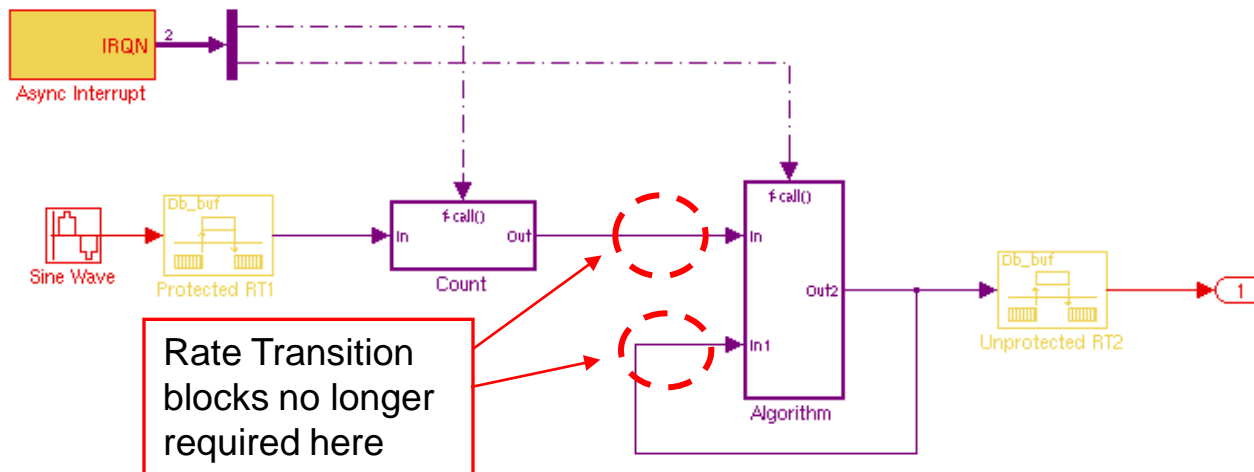
- Rate Transition blocks were required at asynchronous subsystem ports even if rate transition protection was not required.

Benefit

- Simplifies diagram
- Improves code efficiency

Solution

- Allow direct connection between asynchronous subsystems



stddef.h and stdlib.h Included Only When Necessary

Challenge

- Real-Time Workshop always included `stddef.h` and Stateflow always included `stdlib.h`, hampering integration for some environments.

2008a

```

17  #include <stdlib.h>
18  #include <stddef.h>
19  #include "rtwtypes.h"
20  #include "rt_look2d_normal.h"
21  #include "rt_look.h"
22  #include "rt_defines.h"
    
```

Solution

- Inclusion of these files is now controlled by Target Function Library, based on utility function usage.

2008b

```

17  #include "rtwtypes.h"
18  #include "rt_defines.h"
19  #include "rt_look.h"
20  #include "rt_look2d_normal.h"
    
```

Benefit

- Easier integration of generated code because of minimal header file dependencies