

Latest Features in Simulink Fixed Point

October 2008

R2008b

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

Challenge

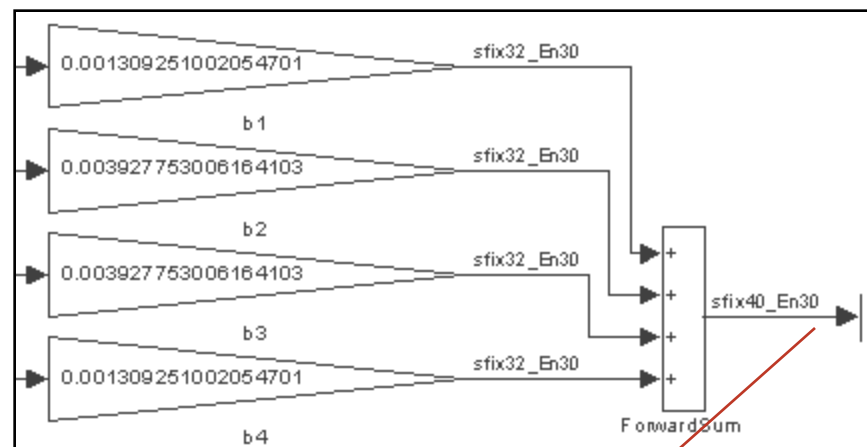
- Code cannot be generated beyond 32 bits for accelerated simulation modes or embedded deployment.
- Simulink simulates up to 128 bits.

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);
```

Embedded MATLAB for Word Sizes up to 128 Bits

Challenge

- Embedded MATLAB was limited to 32 bits.

Solution

- Embedded MATLAB now supports up to 128 bits.

Benefit

- Supports generating efficient code for targets with nonstandard and larger word sizes
- Allows Embedded MATLAB Function block to work with large fixed-point signals

```

1 function y = example_33plus
2     %#eml
3     u1 = fi(2^34,1,45,7);
4     u2 = fi(2^34,1,45,7);
5     y = u1+u2;
    
```

Example multiword code generation for 33+ bits on a Win32 platform

```

eml_r11 = eml_r6;
if(impl_sMultiWordLt(&eml_r5.chunks[0U], 2, &eml_r7.chunks[0U], 2) && impl_s
], 2) && impl_sMultiWordGe(&eml_r10.chunks[0U], 2, &eml_r11.chunks[0U], 2
    eml_r4 = eml_r12;
} else {
    eml_r13 = eml_r0;
    eml_r14 = eml_r6;
    eml_r15 = eml_r0;
    eml_r16 = eml_r6;
    eml_r17 = eml_r4;
    eml_r18 = eml_r6;
    if(impl_sMultiWordGt(&eml_r13.chunks[0U], 2, &eml_r14.chunks[0U], 2) && i
        chunks[0U], 2) && impl_sMultiWordLe(&eml_r17.chunks[0U], 2, &eml_r18
            eml_r4 = eml_r19;
    }
}
eml_r20 = eml_r4;
impl_sMultiWord2sMultiWord(&eml_r20.chunks[0U], 2, &eml_r21.chunks[0U], 3);
eml_r22 = eml_r21;
impl_sMultiWordShl(&eml_r22.chunks[0U], 3, 18, &eml_r23.chunks[0U], 3);
eml_r24 = eml_r23;
    
```

Fixed-Point Advisor Enhancements

Challenge

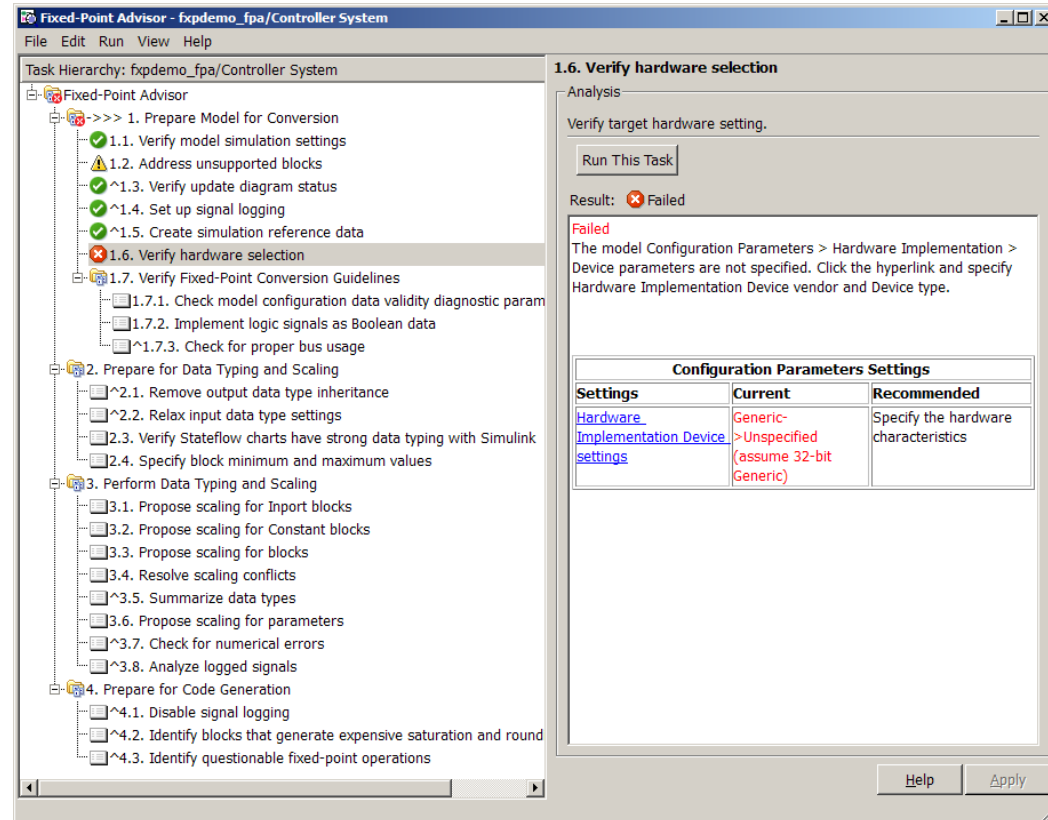
- End users need better documentation and usability.

Solution

- Support the following:
 - Context-sensitive help
 - Results description
 - Table formatting
 - Preserving baseline across runs
 - Linkage to Fixed-Point Tool

Benefit

- Tool is easier to use.
- Information is more accessible.



The screenshot shows the Fixed-Point Advisor interface for a project named 'fxpdemo_fpa/Controller System'. The left pane displays a task hierarchy with the following tasks:

- Fixed-Point Advisor
 - 1. Prepare Model for Conversion
 - 1.1. Verify model simulation settings (Completed)
 - 1.2. Address unsupported blocks (Warning)
 - 1.3. Verify update diagram status (Completed)
 - 1.4. Set up signal logging (Completed)
 - 1.5. Create simulation reference data (Completed)
 - 1.6. Verify hardware selection (Failed)
 - 1.7. Verify Fixed-Point Conversion Guidelines
 - 1.7.1. Check model configuration data validity diagnostic param (Warning)
 - 1.7.2. Implement logic signals as Boolean data (Warning)
 - 1.7.3. Check for proper bus usage (Warning)
 - 2. Prepare for Data Typing and Scaling
 - 2.1. Remove output data type inheritance (Warning)
 - 2.2. Relax input data type settings (Warning)
 - 2.3. Verify Stateflow charts have strong data typing with Simulink (Warning)
 - 2.4. Specify block minimum and maximum values (Warning)
 - 3. Perform Data Typing and Scaling
 - 3.1. Propose scaling for Inport blocks (Warning)
 - 3.2. Propose scaling for Constant blocks (Warning)
 - 3.3. Propose scaling for blocks (Warning)
 - 3.4. Resolve scaling conflicts (Warning)
 - 3.5. Summarize data types (Warning)
 - 3.6. Propose scaling for parameters (Warning)
 - 3.7. Check for numerical errors (Warning)
 - 3.8. Analyze logged signals (Warning)
 - 4. Prepare for Code Generation
 - 4.1. Disable signal logging (Warning)
 - 4.2. Identify blocks that generate expensive saturation and round (Warning)
 - 4.3. Identify questionable fixed-point operations (Warning)

The right pane shows the details for task 1.6, 'Verify hardware selection'. The analysis indicates a failure:

Result: **Failed**

Failed
The model Configuration Parameters > Hardware Implementation > Device parameters are not specified. Click the hyperlink and specify Hardware Implementation Device vendor and Device type.

Configuration Parameters Settings		
Settings	Current	Recommended
Hardware Implementation Device settings	Generic- > Unspecified (assume 32-bit Generic)	Specify the hardware characteristics

Code Optimized to Eliminate Unnecessary Saturations

Challenge

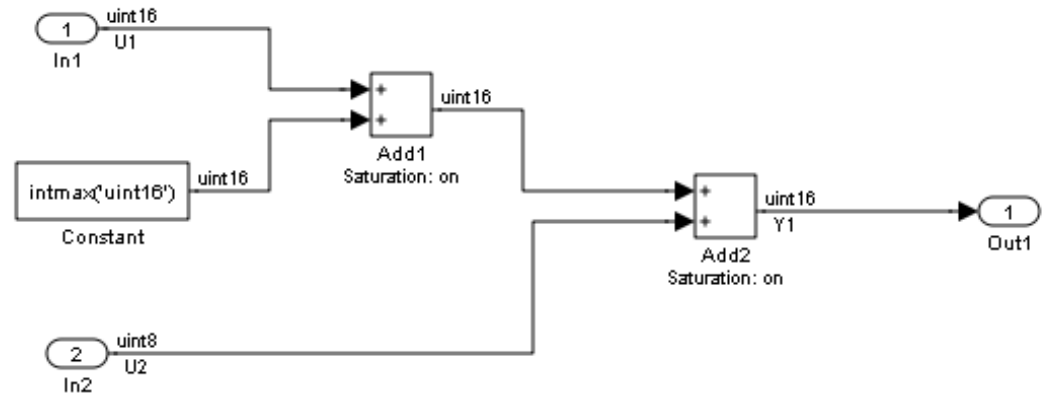
- Saturation logic sometimes contained superfluous code.

Solution

- Remove unnecessary saturation code by performing data range analysis

Benefit

- Reduces RAM and ROM consumption
- Improves execution speed



R2008a

R2008b

```
uint32_T tmp;
uint16_T tmp_0;
tmp = (uint32_T)U1 + 65535U;
if (tmp > 65535U) {
    tmp_0 = MAX_uint16_T;
} else {
    tmp_0 = (uint16_T)tmp;
}
tmp = (uint32_T)tmp_0 +
      (uint32_T)U2;
if (tmp > 65535U) {
    tmp_0 = MAX_uint16_T;
} else {
    tmp_0 = (uint16_T)tmp;
}
Y1 = tmp_0;
```

Y1 = MAX_uint16_T;

For this case, conditions are always true!