RENESAS Tool News

RENESAS TOOL NEWS on December 1, 2004: RSO-M3T-CC32R-041201D

A Note on Using C-Compiler Package M3T-CC32R

Please take note of the following problem in using the M3T-CC32R, the C-compiler package for the M32R family of MCUs:

- On making a function call using a pointer pointing to a function whose type has been converted by a cast operator

---

1. **Versions Concerned**
   M3T-CC32R V.1.00 Release 1 through V.4.30 Release 00

2. **Description**
   When a function call is made using a pointer pointing to a function whose type has been converted by casting, the following error may arise even if both types of each parameter and its argument match with each other:

   error: type of argument does not match with prototype

3. **Conditions**
   This problem occurs if the following conditions are all satisfied:
   (1) The type of a pointer or integer is converted to another pointer pointing to a function by casting, and a function call is made at the same time.
   (2) At least one parameter of the function to be called is array type.

   Example 1: sample1.c
   ```
   ---------------------------------------------------------------------
   short array[3];
   ```

```
void foo1(void *ptr1)

{
   (*(void(*)(int, short arr[]))ptr1)(2,array);
                              /* Conditions (1) and (2) */
}
```

Example 2: sample2.c

```
typedef struct AAA TYPE_A;
char *ptr2;

int foo2(TYPE_A array[][3])
{
   int ans;
   ans = (*(int(*)(TYPE_A arr[][3]))ptr2)(array);
                              /* Conditions (1) and (2) */
   return ans;
}
```

Example 3: sample3.c

```
typedef float TYPE_F[3];
TYPE_F array;

void foo3(void)
{
   (*(void(*)(TYPE_F))0x123400)(array);
                              /* Conditions (1) and (2) */
}
```
---

4. **Workaround**

   When making a function call using a pointer pointing to a function whose type has been converted by casting, once save the result of casting in another pointer variable, and then make a function call using this variable.

   Modification of Example 1:

   ```
   short array[3];
   ```

```
void foo1(void *ptr1)
{
   void (*callptr1)(int, short arr[])
                /* Define a pointer variable "callptr1" */
     = (void (*)(int,short arr[]))ptr1;
                /* Save the cast result in "callptr1" */
   (*callptr1)(2,array);      /* Call a function using "callptr1" */
}
```

Modification of Example 2:

```
typedef struct AAA TYPE_A;
char *ptr2;

int foo2(TYPE_A array[][3])
{
   int ans;
   int (*callptr2)(TYPE_A arr[][3]);
                /* Define a pointer variable "callptr2" */
   callptr2 = (int(*)(TYPE_A arr[][3]))ptr2;
                /* Save the cast result in "callptr2" */
   ans = (*callptr2)(array);  /* Call a function using "callptr2" */
   return ans;
}
```

Modification of Example 3:

```
typedef float TYPE_F[3];
void *ptr3;
TYPE_F array;

void foo3(void)
{
   void (*callptr3)(TYPE_F);
                /* Define a pointer variable "callptr3" */
   callptr3 = (void(*)(TYPE_F))0x123400;
                /* Save the cast result in "callptr3" */
   (*callptr3)(array);      /* Call a function using "callptr3" */
}
```

---

5. **Schedule of Fixing the Problem**

We plan to fix this problem in our next release of the product.
