



Y Explorations Inc.

**Application Note 105: //yx channel_transform Channel to Channel
Assignment Guidelines**

(eXCite 3.2b and later)

February 12, 2008

Author: Pei Zhang, pzhang@yxi.com

Overviews

This document describes guidelines of the channel to channel assignment for eXCite's new pragma `//yx channel_transform`. The basic usage of this pragma can be found in the eXCite documentation.

This document covers the following:

- message/memory channel
- blocking/unblocking channel (only message channel is blocking)
- array/scalar variables using the pragma "`//yx channel_transform`"

In the examples used in this documents, we also use the following two set of channels

- memory channel to nonblocking message channel
- memory channel to blocking message channel

For other types of channels, they have similar situations as the above two scenarios.

In general, the following are similar as memory channel to nonblocking message channel which need a blocking channel for synronization during writing.

- memory channel to memory channel
- blocking message channel to memory channel
- blocking message channel to nonblocking message channel
- nonblocking message channel to memory channel
- nonblocking message channel to nonblocking message channel

The following are similar as memory channel to blocking message channel which don't synronization during writing.

- blocking message channel to blocking message channel
- nonblocking message channel to blocking message channel

One exception is array message channels and array memory channels. Array message channel can be accessed element by element, while memory channel can not. Please refer "Reading scalars, writing arrays - Memory channel to blocking message channel, read/write array element by element" for details.

Basic rules

1. For any variable having `//yx channel_transform` channel pragma
 - Every read will incur a channel read operation
 - Every write will incur a channel write operation

2. For any variable having `//yx channel_transform channel pragma`, always use a buffer and read and write elements to and from the buffer. See below for an example.
3. For non-blocking write, always use a blocking channel for synchronization

Terminology

read array: entire array is read from channel

write array: entire array is written to channel

read array element: array element is read from channel

write array element: array element is written to channel

read scalar: scalar element is read from channel

write scalar: scalar element is written to channel

This document includes the following 4 sections:

- Reading arrays, writing arrays
- Reading arrays, writing scalars
- Reading scalars, writing arrays
- Reading scalar, writing scalars

Reading arrays, writing arrays

Buffers are needed in the form of duplicate array variables to store the array read from channels and written back to channels. In the example below, the buffer is array 'a'.

Examples

- Memory channel to nonblocking message channel, need synchronization

```
//yx channel_transform 1 memory nonblocking read
int x[10];
//yx channel_transform 2 message nonblocking write
int y[10];
//yx channel_transform 101 message blocking write
int sync;
int a[10];

while(1){
    a = x;
    y = a;
    sync = 1;
}
```

Csimgen (please read eXCite document for the usage of csimgen) will transform to:

```
int sync;
int a[10];

//yx channel_transform 101 message blocking write
yx_meschan_init_client(101, 32, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT :9 */
//yx channel_transform 2 message nonblocking write
yx_meschan_init_client(2, 320, 32, 100u, 74, 0); /* NONBLOCKING WRITE TB_ELT :7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 10); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    yx_memchan_read(1, a, 0u); /* hw.c:13 */
    yx_meschan_write(2, a, 0u); /* hw.c:14 */
    sync = 1; /* hw.c:15 */
    yx_meschan_write(101, &sync, 0u); /* hw.c:15 */
    ...
}
```

- Memory channel to blocking message, no synchronization needed

```
//yx channel_transform 1 memory nonblocking read
int x[10];
//yx channel_transform 2 message blocking write
int y[10];
int a[10];

while(1){
    a = x;
    y = a;
}
```

Csimgen results in:

```
int a[10];
//yx channel_transform 2 message blocking write
yx_meschan_init_client(2, 320, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT :7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 10); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    yx_memchan_read(1, a, 0u); /* hw.c:11 */
    yx_meschan_write(2, a, 0u); /* hw.c:12 */
}
```

Notes

- You may need different buffer array variables for read and write channels if the array is different for the read and write.

Reading arrays, writing scalars

Buffers are needed to store the array read from the channel.

Examples

- Memory channel to nonblocking message channel, need synchronization

```
//yx channel_transform 1 memory nonblocking read
int x[10];
//yx channel_transform 2 message nonblocking write
int y;
//yx channel_transform 101 message blocking write
int sync;
int a[10];

while(1){
    a = x;
    y = a[5];
    sync = 1;
}
```

Csimgen results are:

```
//yx channel_transform 2 message nonblocking write
int y;
//yx channel_transform 101 message blocking write
int sync;
int a[10];
int yx_est1;
//yx channel_transform 101 message blocking write
yx_meschan_init_client(101, 32, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT :9 */
//yx channel_transform 2 message nonblocking write
yx_meschan_init_client(2, 32, 32, 100u, 74, 0); /* NONBLOCKING WRITE TB_ELT :7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 10); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    yx_memchan_read(1, a, 0u); /* hw.c:13 */
    yx_est1 = a[5]; /* hw.c:14 */
    y = yx_est1; /* hw.c:14 */
    yx_meschan_write(2, &y, 0u); /* hw.c:14 */
    sync = 1; /* hw.c:15 */
    yx_meschan_write(101, &sync, 0u); /* hw.c:15 */
}
```

```
}
```

- Memory channel to nonblocking message channel, need synchronization

```
//yx channel_transform 1 memory nonblocking read
int x[10];
//yx channel_transform 2 message blocking write
int y;
int a[10];

while(1){
    a = x;
    y = a[5];
}
```

Csimgen results are:

```
int y;
int a[10];
int yx_est1;
tb_set_fpu_single_precision(); /* :0 */
//yx channel_transform 2 message blocking write
yx_meschan_init_client(2, 32, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT:7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 10); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    yx_memchan_read(1, a, 0u); /* hw.c:11 */
    yx_est1 = a[5]; /* hw.c:12 */
    y = yx_est1; /* hw.c:12 */
    yx_meschan_write(2, &y, 0u); /* hw.c:12 */
}
```

Notes

- In above example, since array “x” is defined as an array and passed in one piece through channel 1, ‘x’ should not be accessed anywhere in the code by element . This restriction only applies when the entire array is passed through a channel. This restriction will be relaxed in future versions. The following is currently illegal code and not supported.

```
//yx channel_transform 1 memory nonblocking read
int x[10];
int y;
y = x[5]
```

In this case, please use an array buffer first to get the whole array , then access the individual element of the buffer array.

Reading scalars, writing arrays

Buffers are needed to store the array written to channels.

Examples

- Memory channel to nonblocking message channel, need synchronization

```
//yx channel_transform 1 memory nonblocking read
int x;
//yx channel_transform 2 message nonblocking write
int y[10];
//yx channel_transform 101 message blocking write
int sync;
int i, a, b[10];

while(1){
    a = x;
    for (i=0; i< 10; i++)
        b[i] = a+i;
    y = b;
    sync = 1;
}
```

Csimgen results in:

```
//yx channel_transform 1 memory nonblocking read
int x;
//yx channel_transform 101 message blocking write
int sync;
int i;
int a;
int b[10];
//yx channel_transform 101 message blocking write
yx_meschan_init_client(101, 32, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT :9 */
//yx channel_transform 2 message nonblocking write
yx_meschan_init_client(2, 320, 32, 100u, 74, 0); /* NONBLOCKING WRITE TB_ELT :7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 1); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    yx_memchan_read_elt(1, &x, 0u, 0); /* hw.c:13 */
```

```

    a = x; /* hw.c:13 */
    for(i = 0; i < 10; i = i + 1) b[i] = a + i; /* hw.c:15 */ /* hw.c:14 */
    yx_meschan_write(2, b, 0u); /* hw.c:16 */
    sync = 1; /* hw.c:17 */
    yx_meschan_write(101, &sync, 0u); /* hw.c:17 */
}

```

- Memory channel to blocking message channel, no synchronization needed

```

//yx channel_transform 1 memory nonblocking read
int x;
//yx channel_transform 2 message blocking write
int y[10];
int a, b[10];
int i;

while(1){
    a = x;
    for (i=0; i < 10; i++)
        b[i] = a+i;
    y = b;
}

```

Csimgen results in:

```

//yx channel_transform 1 memory nonblocking read
int x;
int a;
int b[10];
int i;
tb_set_fpu_single_precision(); /* :0 */
//yx channel_transform 2 message blocking write
yx_meschan_init_client(2, 320, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT :7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 1); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    yx_memchan_read_elt(1, &x, 0u, 0); /* hw.c:12 */
    a = x; /* hw.c:12 */
    for(i = 0; i < 10; i = i + 1) b[i] = a + i; /* hw.c:14 */ /* hw.c:13 */
    yx_meschan_write(2, b, 0u); /* hw.c:15 */
}

```

- Memory channel to blocking message channel, read/write array element by element

Another way to write a message blocking array is element by element. However, this method has some issues.

```
//yx channel_transform 1 memory nonblocking read
int x;
//yx channel_transform 2 message blocking write
int y[10];
int i;

while(1){
    for (i=0; i < 10; i++)
        y[i] = x+i;
}
```

Csimgen results in:

```
//yx channel_transform 1 memory nonblocking read
int x;
int i;
int yx_est1;
int yx_est2;
//yx channel_transform 2 message blocking write
yx_meschan_init_client(2, 32, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT :7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 1); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    for(i = 0; i < 10; i = i + 1){
        yx_memchan_read_elt(1, &x, 0u, 0); /* hw.c:12 */
        yx_est1 = x; /* hw.c:12 */
        yx_est2 = yx_est1 + i; /* hw.c:12 */
        yx_meschan_write(2, &yx_est2, 0u); /* hw.c:12 */
    } /* hw.c:11 */ /* hw.c:11 */
}
```

In the software, we have to do the following:

```
#include "typedef_io.h"

register int j;
int x, y;
yx_io_init();
```

```
x = 63;
for (j=0; j< 10; j++) {
    yx_write_x(&x); /* (1) */
    yx_read_y(&y); /* (1) */
}
```

Please refer to the eXCite documentation for **//yx generate_io** for the details of **yx_write_x/yx_read_y**

When using this method, on hardware side the array “y” is accessed element by element after Csimgen transfer, a scalar variable has to be used on the software side to guarantee correct communication with array “y” on the hardware side. So on the hardware side, “y” is defined as array, but the corresponding part on software side is defined as scalar.

Another limitation is this method is only works for message channel, since for we need index for memory channel element access, but the scalar variable memory channel on software side can not provide this index.

Reading scalars, writing scalars

Buffer variables are not needed. Remember that every variable read/write will incur a channel access

Examples

- Memory channel to nonblocking message channel, need synchronization

```
//yx channel_transform 1 memory nonblocking read
int x;
//yx channel_transform 2 message nonblocking write
int y;
//yx channel_transform 101 message blocking write
int sync;

while(1){
    y = x;
    sync = 1;
}
```

csimgen results:

```
//yx channel_transform 1 memory nonblocking read
int x;
//yx channel_transform 2 message nonblocking write
int y;
//yx channel_transform 101 message blocking write
int sync;
//yx channel_transform 101 message blocking write
yx_meschan_init_client(101, 32, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT :9 */
//yx channel_transform 2 message nonblocking write
yx_meschan_init_client(2, 32, 32, 100u, 74, 0); /* NONBLOCKING WRITE TB_ELT :7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 1); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    yx_memchan_read_elt(1, &x, 0u, 0); /* hw.c:12 */
    y = x; /* hw.c:12 */
    yx_meschan_write(2, &y, 0u); /* hw.c:12 */
    sync = 1; /* hw.c:13 */
    yx_meschan_write(101, &sync, 0u); /* hw.c:13 */
}
```

- Memory channel to blocking message channel, no synchronization needed

```
//yx channel_transform 1 memory nonblocking read
int x;
//yx channel_transform 2 message blocking write
int y;

while(1){
    y = x;
}
```

csimgen results:

```
//yx channel_transform 1 memory nonblocking read
int x;
//yx channel_transform 2 message blocking write
int y;
tb_set_fpu_single_precision(); /* :0 */
//yx channel_transform 2 message blocking write
yx_meschan_init_client(2, 32, 32, 100u, 73, 0); /* BLOCKING WRITE TB_ELT :7 */
//yx channel_transform 1 memory nonblocking read
yx_memchan_init_client(1, 1, 78u, 0, 32, 1, 1); /* NONBLOCKING READ WRITE TB_ELT:5 */
while(1){
    yx_memchan_read_elt(1, &x, 0u, 0); /* hw.c:10 */
    y = x; /* hw.c:10 */
    yx_meschan_write(2, &y, 0u); /* hw.c:10 */
}
```

Notes:

- Though a buffer is not needed for the scalar variable read/write, we do recommend using buffer variables if scalar variables are read/write multiple times in the C codes to minimize bus/channel I/O cycles.