How to use notifications over interfaces

version	1.1.1
scope	Example. This code is provided as example code for a user to base their code on.
description	How to use notifications over interfaces
boards	Unless otherwise specified, this example runs on the SliceKIT Core Board, but can easily be run on any XMOS device by using a different XN file.

Sometimes the server end of an interface needs to signal information to the client end. However, usually the client end initiates communication.

Notifications provide a way for the server to contact the client independently of the client making a call. It can raise a signal and then carry on processing.

Within the interface declaration, a notification function can be declared with the `[[notification]]` attribute.

```
[[notification]] slave void data_ready(void);
```

The function is declared as `slave` to indicate the direction of communication is from the server end to the client end. In other words, the server will call the function and the client will respond. Notification functions must take no arguments and have a `void` return type.

Once the server raises a notification, it triggers an event at the client end of the interface. However, repeatedly raising the notification has no effect until the client clears the notification. This can be done by marking one or more functions in the interface with the `[clears_notification]` attribute.

```
[[clears_notification]] int get_data();
```

The client will then clear the notification whenever it calls that function.

The server end of the interface can call the notification function to notify the client end. One important property of notifications is that they will not block and the server can continue doing work.

```
c.data_ready();
// The above call is non-blocking , so the task carries on
printf("task1: Sent notification\n");
```

After calling `data_ready`, calling it again will have no effect (i.e. the signal can only be raised once). The server end can then carry on with processing, including receiving messages from the client end of the same interface connection.

```
// Wait for some incoming messages
for (int i = 0;i < 2; i++) {
  select {
   case c.msg(int x):
     printf("task1: Received data message %d\n",x);
     break;
   case c.get_data() -> int return_value:
     printf("task1: Received response to notification\n");
     return_value = data;
     break;
  }
}
```

After the `get_data` call has been received, this task could re-notify the client.

The client end of the interface can make calls as normal, but can also select upon the notification from the server end of the interface.

```
void task2(client interface if1 c)
{
  c.msg(5);
  select {
  case c.data_ready():
    int x = c.get_data();
    printf("task2: Got data %d\n",x);
    break;
  }
}
```