xCORE trycatch library

A library providing a way to handle hardware and thrown exceptions.

By default, exception are caught in a trap handler that halts the tile, awaiting external intervention. This library handles exceptions by unwinding the stack to the most recently registered trycatch block. The catch clause is then executed.

This library can not be used to implement commit-rollback semantics. Resources are not returned to their pre-try state. The user is required to handle exception safety of global memory and other resources.

Features

- Catches hardware generated and thrown exceptions
- User may `throw` exceptions.
- Trycatch blocks may be nested.

Limitations

- xCORE resources allocated inside the trycatch block may not be freed if an exception is raised.
- If an exception is raised the values of local variables changed inside the trycatch block are indeterminate.
- If the code inside the trycatch block spawns task onto additional logical cores, exceptions on these logical cores will not be caught.
- The compiler may remove code that has no other side effects beyond raising an exception.

Software version and dependencies

This document pertains to version 1.0.0 of this library. It is known to work on version 14.2.1 of the xTIMEcomposer tools suite, it may work on other versions.

The library does not have any dependencies (i.e. it does not rely on any other libraries).

1 Usage

This library contains macros that allow you to handle hardware and thrown exceptions raised on the current logical core.

All functions can be accessed via the `trycatch.h` header:

```
#include "trycatch.h"
```

You will also have to add `lib_trycatch` to the USED_MODULES field of your application Makefile.

The library provides three macros:

```
#define TRY
#define CATCH(exception)
#define THROW(exception)
```

and a data structure:

```
typedef struct exception_t {
  unsigned type; // Exception type.
  unsigned data; // Exception data.
} exception_t;
```

The TRY macro must be immediately followed by a CATCH macro as follows:

```
exception_t exception;
TRY { ... } CATCH(exception) { ... }
```

If an exception is raised, execution jumps to the code inside the catch block. The operand of the CATCH macro is populated with information about the raised exception. The catch block is not executed if no exception is raised.

An exception may be raised using the THROW macro as follows:

```
exception_t e = {256,0};
THROW(e);
```

The TRY, CATCH and THROW macros are implemented using setjmp() and longjmp() and have the limitations specified in the Limitations section.

1.1 Example

If we have a function that may cause an exception to fire:

```
void may_exception_func();
```

and a function that calls it, that must catch those exceptions:

```
void no_exception_func();
```

we must call `may_exception_func` in a trycatch block, providing a thread safe `exception_t` variable for this trycatch block. The easiest way to do this is to make it an auto variable (on the stack):

```
exception_t exception;
```

The TRY clause is followed immediately by a statement (or statement block) that is executed:

```
TRY {
  may_exception_func();
}
```

and this followed immediately by CATCH and a statement (or statement block) that is executed only if an exception was caught:

```
CATCH (exception) {
  debug_printf("exception: type=%d data=%d\n",
                exception.type, exception.data);
}
```

Here is a complete example (build using -O0 to make sure the divide happens):

```
// xs1.h pulls in 'XS1_ET_' constants.
#include <xs1.h>
#include "trycatch.h"
#include "debug_print.h"

int divide(int dividend, int divisor) {
  // Uncomment to send an unexpected exception.
  // exception_t e = {256,0};
  // THROW(e);
  return dividend / divisor;
}

int main() {
  exception_t exception;
  TRY {
    int result = divide(42, 0);
    debug_printf("Unexpected success: %d\n", result);
  }
  CATCH (exception) {
    if (exception.type == XS1_ET_ARITHMETIC) {
      debug_printf("Divide by zero caught\n");
    } else {
      debug_printf("Unexpected exception: type=%d data=%d\n",
                    exception.type, exception.data);
    }
  }
  return 0;
}
```

2 API

2.1 Supporting types

Type	exception_t
Description	Structure describing an exception. The following hardware exception types are defined by xs1.h: • XS1_ET_NONE • XS1_ET_LINK_ERROR • XS1_ET_ILLEGAL_PC • XS1_ET_ILLEGAL_INSTRUCTION • XS1_ET_ILLEGAL_RESOURCE • XS1_ET_LOAD_STORE • XS1_ET_ILLEGAL_PS • XS1_ET_ARITHMETIC • XS1_ET_ECALL • XS1_ET_RESOURCE_DEP • XS1_ET_KCALL Please see 'The XMOS XS* Architecture' document for further details. Runtime errors in software are trapped using an XS1_ET_ECALL hardware exception. Values greater than 255 should be used for a 'thrown' exception type.

Fields	`unsigned type`
	Exception type.
	
	`unsigned data`
	Exception data.

2.2 Macros

Macro	**TRY**

Description	Macro to execute a block of code catching any raised exceptions.
	The TRY macro must be immediately followed by a CATCH macro as follows: If an exception is raised, execution jumps to the code inside the catch block. The operand of the CATCH macro is populated with information about the raised exception. The catch block is not executed if no exception is raised.
	`exception_t exception;`
	`TRY { ... } CATCH(exception) { ... }`
	The TRY and CATCH macros are implemented using `setjmp()` and `longjmp()` and have the following limitations:
	• xCORE resources allocated inside the TRY block may not be freed if an exception is raised.
	• If an exception is raised the values of local variables changed inside the TRY block are indeterminate.
	• If the code inside the TRY block spawns task onto additional logical cores, exceptions on these logical cores will not be caught.
	• The compiler may remove code that has no other side effects beyond raising an exception.

Macro	**CATCH**
Description	Macro for catching an exception. This must be used in conjunction with the TRY macros, see documentation of TRY for more details.
Parameters	`exception` the unique exception variable for this TRY-CATCH block

Macro	**THROW**

Description	Macro for throwing an exception.
	This must only be called from within an active TRY-CATCH block as follows:
	`void mayThow(void) {`
	`  exception_t e = {256,0};`
	`  THROW(e);`
	`}`
	`void noThow(void) {`
	`  exception_t e;`
	`  TRY`
	`    mayThow();`
	`  CATCH(e)`
	`    assert(e.type == 256);`
	`}`

Parameters	`exception` the exception to be thrown

APPENDIX A - Known Issues

No known issues.

APPENDIX B - lib_trycatch change log

B.1 1.0.0

- Initial release