

INSTALLATION INSTRUCTIONS.....	2
ASSUMPTION OVER THE SOURCE PROGRAM	2
ABSTRACTIONS OVER THE SOURCE PROGRAM.....	2
PROCESS ABSTRACTION.....	2
CHANNEL ABSTRACTION.....	3
DATA TYPE ABSTRACTIONS	4

Installation Instructions

Simply put the “spin2core” executable file in your designated library.

Operate the “spin2core” file from the directory where your source file is found.

Example: If your spin2core file is in the directory Spin2Core and your spin file is in the Spin2Core/example directory, go to the example directory and execute the following line (in Unix):

```
../spin2core my_file.spin
```

Assumption over the source program

1. The source program is a valid spin program.
2. Spin2Core does not support labels and GOTO instructions, it will not translate them.
3. The abstraction of the channel is not fully implemented: it does translate a channel to a module, and tries to partially synchronized the process-module with the channel-module, but the result is not a legal core file...

Abstractions over the source program

Process abstraction

Each process will be translated to a module, where the module’s name will be the same as the process’s name.

The module will have a local variable, simulating the program counter of the process.

Each statement will be translated into transition where its enable condition will be the PC, and it will assign the PC with the value of the following statement.

Example (for the PC abstraction):

```
TRANS some_trans
    enable: pc = 6 . translated_condition
    assign: pc' := 7; translated_statement;
TRANS following_trans
    enable: pc = 7 . translated_condition
    assign: pc' := 8; translated_statement;
```

Channel abstraction

Each channel will be translated to a module, where the module's name will be the same as the channel's name.

The module will have a local variable, counting the number of the messages in the channel in any given time.

The module will also have buffers, simulating the cyclic buffer of the channel. For each element of the message, there will be a different buffer.

Two pointers (integer variable) will indicate the place in which new message will be inputted, and the next message that will be outputted.

Example:

```
chan testing = [15] of {mtype, int, bit}
```

will be translated to a module with 6 parameters- two parameters for each data type of the message, one for receiving and the other for sending.

It will have only two transitions: put (for receiving a message from user and putting in the buffer) and get (the opposite).

Any process, using the channel will have a receive/send transition that will be partially synchronized with the channel-module, and it will have the corresponding parameters for the message's data types.

Data type abstractions

BIT

Will be translated as Boolean.

BYTE

Will be translated as range of numbers from -2^8 to 2^8-1

SHORT

Will be translated as range of numbers from -2^{16} to $2^{16}-1$

MTYPE

Will be translated as Enumerate.

STRUCT

Each field will be translated as separate variable.

All other types will be translated directly.