Occam Run-time Model Specification

Andy Whitlow

SW-0064-4

APPROVED 11 June, 1990

Contents

1 Introduction

This document describes the runtime environment for the product Occam compiler, **oc**.

2 Scheduling

Processes in occam may have one of two priorites, high or low. A high priority process will be executed in preference to a low priority process if both are active, so that the low priority process will be interrupted. A high priority process is initiated by using the PRI PAR construct which takes two processes, the first of which is to be executed at high priority. PRI PAR constructs may not be nested.

Scheduling in occam is achieved using the transputer's scheduler. This scheduler enables any number of concurrent processes to be executed together, sharing processor time.

At any time a concurrent process may be :

1 **Active** - being executed or waiting to be executed.

2 **Inactive** - ready to communicate or waiting until a specified time.

The scheduler maintains lists of processes. These lists are implemented using special registers pointing to the list head and list tail in workspace. Each process uses special below workspace slots (see section 9.1) to chain the workspaces in the list together.

There exist two compiler predefines which affect scheduling. They are:

1 **CAUSEERROR()** : This inserts a **seterr** instruction into the program. An additional **stoperr** is inserted if the program is compiled in STOP or UNIVERSAL mode.

2 **RESCHEDULE()** : This inserts enough instructions into the program to cause the current process to be placed onto the back of the process queue.

For more information on scheduling see [1].

3 Communication

Communication between occam processes is achieved by means of channels. occam communication is point-to-point, synchronised and unbuffered.

Communication takes place when both the inputting and outputting processes are ready. Consequently, the process which first becomes ready must wait until the second one is also ready.

For more information on communication see [1].

4 Error Modes

oc supports 3 levels of error handling. These are:

`Halt` In this mode an error causes the transputer to halt. The transputers **`HaltonError`** flag should be TRUE.

`Stop` In this mode an error in a process will cause that particular process to stop. The transputers **`HaltonError`** flag should be false.

`Universal` This mode is a generic error mode that can be used with modules compiled in Halt or Stop mode. It behaves like Halt or Stop mode depending on the state of the transputer **`HaltonError`** flag. A module compiled in Halt or Stop mode may call a module compiled in Universal mode but NOT vice versa. A module compiled in Universal mode may only call another Universal module.

`Undefined` mode is no longer directly supported, however it can be implemented by use of a compiler pragma or switch which disables run-time checks.

For more information on the occam error modes see [2].

5 Type Mapping

This section defines all the occam types and how they are represented in the processors.

All items are word aligned and are little-endian.

5.1 32 Bit Machine

The occam types are represented on a 32-bit transputer as described in the following table.

BOOL	Represented in a word in which the lowest bit is significant, the upper bits are zero.
BYTE	Represented in a word in which the lower eight bits are significant, the upper bits are zero.
INT INT32	Represented in a word in which all 32 bits are significant using twos complement form.
INT16	Represented in a word in which the lower sixteen bits are significant using twos complement form, the upper bits are undefined.
INT64	Represented in two words in which all 64 bits are significant using twos complement form.
REAL32 REAL64	Represented in a word, in IEEE single-precision format. Represented in two words, in IEEE double-precision format.
TIMER	A timer occupies no storage.
CHAN	A channel is implemented as a pointer to a channel word. Previous compilers implemented a channel as a word in memory containing the channel value. This implementation can be obtained by use of a compiler switch.
PORT	Ports are represented the same way as the datatype for which they are a port.

5.2 16 Bit Machine

The occam types are represented on a 16-bit transputer as described in the following table.

BOOL	Represented in a word in which the lowest bit is significant, the upper bits are zero.
BYTE	Represented in a word in which the lower eight bits are significant, the upper bits are zero.
INT INT16	Represented in a word in which all 16 bits are significant using twos complement form.
INT32	Represented in two words in which all 32 bits are significant using twos complement form.
INT64	Represented in four words in which all 64 bits are significant using twos complement form.
REAL32 REAL64	Represented in two words, in IEEE single-precision format. Represented in four words, in IEEE double-precision format.
TIMER	A timer occupies no storage.
CHAN	A channel is implemented as a pointer to a channel word. Previous compilers implemented a channel as a word in memory containing the channel value. This implementation can be obtained by use of a compiler switch.
PORT	Ports are represented the same way as the datatype for which they are a port.

5.3 Aggregate Types

Arrays are packed. The following table denotes the representation of each element in an array for a given type on a 32 bit transputer:

BOOL	Each element is represented as a byte in which the lowest bit is significant, the upper bits are zero.
BYTE	Each element is represented in a byte in which all the bits are significant, the upper bits are zero.
INT INT32	Each element is represented in a word in which all 32 bits are significant using twos complement form.
INT16	Each element is represented in two bytes in which all 16 bits are significant using twos complement form.
INT64	Each element is represented in two words in which all 64 bits are significant using twos complement form.
REAL32 REAL64	Each element is represented in a word, in IEEE single-precision format. Each element is represented in two words, in IEEE double-precision format.
TIMER	An array of timers occupies no storage.
CHAN	Each element is represented as a pointer to a word in memory containing the channel contents. Previous compilers implemented each element as a word in memory containing the channel value. This implementation can be obtained by use of a compiler switch.
PORT	Each element is represented in the same way as the array element representation for the datatype for which it is a port.

The following table denotes the representation of each element in an array for a given type on a 16 bit transputer:

BOOL	Each element is represented as a byte in which the lowest bit is significant, the upper bits are zero.
BYTE	Each element is represented in a byte in which all the bits are significant, the upper bits are zero.
INT INT16	Each element is represented in a word in which all 16 bits are significant using twos complement form.
INT32	Each element is represented in two words in which all 32 bits are significant using twos complement form.
INT64	Each element is represented in four words in which all 64 bits are significant using twos complement form.
REAL32 REAL64	Each element is represented in two words, in IEEE single-precision format. Each element is represented in four words, in IEEE double-precision format.
TIMER	An array of timers occupies no storage.
CHAN	Each element is represented as a pointer to a word in memory containing the channel contents. Previous compilers implemented each element as a word in memory containing the channel value. This implementation can be obtained by use of a compiler switch.
PORT	Each element is represented in the same way as the array element representation for the datatype for which it is a port.

Protocol tags are represented by 8-bit values. The compiler allocates such values from 0(BYTE) upwards in order of declaration .

5.4 Virtual Channels

Virtual channels are represented as a pointer to an area of memory which is used for the communication. Communication on a virtual channel is achieved by passing the virtual channel pointer to the virtual channel communication routines in a similar fashion to a normal communication where the address of a channel word is used as an operand to one of the channel communication instructions.

Currently a virtual channel is distinguished from a normal channel by means of the bottom bit of its address. If this bit is set then the channel is a virtual channel.

For more information on virtual channels see [3].

5.5 Timers

Timers make use of the transputer's clock. The low priority clock increments at a rate of 15625 ticks per second (1 tick = 64 microseconds). The high priority clock increments at a rate of 1000000 ticks per second (1 tick = 1 microsecond).

6 Parameter Passing

Parameters are divided into VAL and non-VAL parameters. These have different semantics and may be passed differently.

6.1 VAL parameters

Scalar values that fit within the wordlength of the target machine are represented as items one word long containing the value of the parameter.

In the case of BYTE and BOOL, the value is found in the low-order byte of the word and the high order bytes are guaranteed to be zero.

In the case of an INT16 parameter on a 32-bit processor, the value resides in the low-order 2 bytes of the word. The high-order bytes are undefined.

If the parameter is a primitive type that will not fit into a processor word, then it is passed as a pointer to the actual value.

If the parameter is an array, it may have some of its strides undefined in the source. In this case, extra parameters are passed containing the integer values of the missing strides. These parameters are placed immediately after the address of the first element of the array, which constitutes the parameter representation. They appear in the same order as the missing strides appear in the source.

Timers, channels and ports can never be VAL parameters.

6.2 Non-VAL parameters

Since these parameters change the actual parameters passed, the formal parameters are always represented as pointers to the actuals (except for timers - see below).

If the parameter is an array, then it is treated the same way as a VAL parameter with any missing strides following the address of the first element.

If the parameter is a timer, it occupies no storage and so no parameter slot is reserved for it.

7 Calling Sequence

This section describes the state on entry to the called routine.

7.1 Registers

Areg is undefined. In fact, if the occam code calls a library routine, then the *call* is done by means of a call instruction to a stub, which then does a *j* to the true entry point. The linker patches the code where this *j* is to provide the correct offset. This *j* could destroy the register stack contents because of the possibility of a timeslice occuring, in this case all three registers would be undefined. (Calls to procedures and functions defined within the same compilation module cannot timeslice).

In the T800, the floating point registers are never assumed to have any values.

Iptr addresses the first instruction of the invoked procedure or function.

Wptr addresses the invocation stack frame (see next section). It must be word aligned.

7.2 Invocation stack

The invocation stack at entry to a procedure or function is addressed by non-negative offsets from Wptr. Negative offsets are in the free (unused) part of the invocation stack.

```
word    |               |  (high addresses)
offset  |               |
        |---------------|
  n     |  parm  n      |  (last parameter passed in)
        |               |
        :               :
        |               |
        |---------------|
  4     |  parm  4      |  (first parm stored by caller)
        |---------------|
  3     |  parm  3      |  (C reg as saved by call instruction)
        |---------------|
  2     |  parm  2      |  (B reg as saved by call instruction)
        |---------------|
  1     |  parm  1      |  (A reg as saved by call instruction)
        |---------------|
  0     |  Iret         |  (return address if call used)
        |---------------|  <---  Wptr
 -1     |  free         |
        :               :
        |---------------|  (top of stack)
```

7.3 Iret

In all cases, the value of Iret has been stored before the first instruction of the called proc is executed.

7.4 Parameters

The parameters to the procedure are found at successive words from Wptr+ k (words), where 1 $<=$ k $<=$ 3. The first two parameters are optionally the static link and vector space address (In some cases the vector space address may be the last parameter - see below). Each parameter occupies just one word. The source code parameters are placed in the appropriate number of parameter words (see below for details) in the lexical order in the source program.

The pointers to indicate the FUNCTION return value positions (if there are any) come before all the source parameters (see below).

In some cases, the procedure needs the address of the outer level stack frame (the static link). In this case, it is the first parameter to the procedure or function. (Note that this is never true for externally visible functions).

If the procedure being called allocates arrays in the vector space, the current value of the vector space pointer is also passed in as a parameter. For calls to externally visible procedures or functions the parameter is the last one of all. Otherwise, the parameter is the first following the static link (if any). This parameter is optional.

8 Function return state

8.1 Registers

The Areg, Breg and Creg contain the first three word scalar values in that order; except in the case of a floating point transputer, e.g. the T800, returning a single floating value (either REAL32 or REAL64).

For the floating point transputers only, if there is a single floating point return value (and no other return values at all), then the value is returned in FAreg. Otherwise, floating values are returned in the same manner as other return values, i.e. in an integer register if they fit, or via a pointer for values larger than a word (see below). In all cases, FBreg and FCreg have undefined contents.

Wptr has the same value as before the `call` instruction which was used to enter the routine.

Iptr addresses the instruction following the call.

8.2 Return values

Return values are found in various places depending on their datatype and the number of them.

The first three values of a word scalar type are returned in the Areg, Breg and Creg respectively.

For the floating point transputers only, if there is a single return value, and it is of a floating point type (REAL32, REAL64), then it is returned in FAreg. If there are at least two return values, then all the floating type values are returned like any other values, ie. REAL32 are returned in an integer register, if they fit, and REAL64 values are returned in slots as described next.

All other values are returned in slots, whose addresses were passed as extra parameters to the function. These extra parameters are added at the start of the list of normal parameters in the order defined in the list in the source code.

9 Memory Allocation

The memory map of the default occam run-time system is as follows:

```
MOSTNEG INT + IBOARDSIZE --> +---------------+
                             |               |
                             |     FREE      |
                             |    MEMORY     |
                             |               |
                             +---------------+
                             |    VECTOR     |
                             |    SPACE      |
                             +---------------+
                             |               |
                             |     CODE      |
                             |               |
                             +---------------+
                             | SCALAR SPACE  |
                             | (WORKSPACE)   |
                MemStart --> +---------------+
                             |   RESERVED    |
             MOSTNEG INT --> +---------------+
```

Note that a separate workspace (stack_memory) may be allocated by the bootstrap tool. This extra workspace is positioned at **MemStart**, below the occam workspace. (see section 10 for more details).

9.1 Workspace Allocation

Scalar variables are placed in the workspace area and addressed via the Wptr. The compiler will allocate them and supply the debugger with information on where each variable is found. Variables in workspace are allocated on the basis of their estimated usage. The most used variables are given smaller offsets in workspace.

There are a number of different situations in which workspace is allocated. They are as follows:

- **Called Routines** : The workspace for a called routine is allocated from higher to lower addresses, i.e. it grows down memory. This means that the workspace for a called procedure is nearer **MOSTNEG INT** than the workspace for the caller.
- **PAR or PRI PAR constructs** : In a PAR or PRI PAR construct the last textually defined process is allocated the lowest addressed workspace.
- **Replicated PAR** : In a replicated PAR construct the instance with the highest replication count is allocated the lowest workspace address.

There are also situations where special workspace slots are used for scheduling, communication and timer input. These special slots have small negative offsets from the Wptr. A small number of instructions use the slot (Wptr + 0) as an extra register. These instructions are **outword**, **outbyte**,

postnormsn and instructions to implement **ALT**.

The following table describes the situations where special slots or workspace zero will be used by a process:

Process Type	**Special Slots**	**Wptr + 0**
Process with no I/O	2 words	
Process with only unconditional I/O using **in** and **out**	3 words	
Process with only unconditional I/O using **outbyte** or **outword**	3 words	√
Process with alternative input	3 words	√
Process with timer input	5 words	
Process with alternative timer input	5 words	√

For more information on special workspace slots see [2].

9.2 Vectorspace Allocation

If vectorspace is enabled then objects larger than 8 bytes (apart from those explicitly placed in workspace) are allocated in a separate data space. Objects in vectorspace are accessed via a vectorspace pointer which is passed as a parameter to each routine which uses vectorspace or has descendants which use vectorspace.

The allocation of vectorspace is similar to that of workspace. The difference being that vectorspace grows upward in memory, i.e. the data space for a called procedure is at a higher address than the data space of its caller.

10 Initialisation

10.1 Occam Program Interface

The outermost procedure of an occam program, which is to be booted using the icollect tool, must conform to one of two possible procedure declarations:

- Default occam interface:

  This represents the simplest case, where the memory map as shown above applies.

  The program interface:

```
#INCLUDE "hostio.inc"
PROC  program ( CHAN OF SP from.link,
                CHAN OF SP to.link,
                []INT free.memory)
```

  Note that the program name can be any valid occam identifier.

  The parameters are as follows:

from.link : The input channel of the transputer link down which the transputer was booted.

to.link : The output channel of the transputer link down which the transputer was booted.

free.memory : An array representation of the unallocated memory, i.e, **free.memory** points to the first location beyond vector space and has the same number of elements as there are words in the unallocated memory space.

- occam interface when access to stack.memory workspace is required:

  In this case the memory map may have an extra area beginning at **MemStart** as follows:

```
:               :
|      CODE      |
|                |
+----------------+
|  SCALAR SPACE  |
|  (WORKSPACE)   |
+----------------+
|  STACK.MEMORY  |
|   WORKSPACE    |
+----------------+ <--- MemStart
|    RESERVED    |
+----------------+ <--- MOSTNEG INT
```

  The program interface:

```
#INCLUDE "hostio.inc"
PROC  program ( CHAN OF SP from.link,
                CHAN OF SP to.link,
                []INT free.memory,
                []INT stack.memory)
```

  Note that the program name can be any valid occam identifier.

  The parameters are as follows:

from.link : The input channel of the transputer link down which the transputer was booted.

to.link : The output channel of the transputer link down which the transputer was booted.

free.memory : An array representation of the unallocated memory, i.e, **free.memory** points to the first location beyond vector space and has the same number of elements as there are words in the unallocated memory space.

stack.memory : An array representation of the non-occam workspace. i.e, **stack.memory** points to the **Memstart** and has the same number of elements as there are words in the non-occam workspace.

  The non-occam workspace is allocated by the bootstrap tool if requested by the user.

- occam interface for use with the new configuration system.

The program interface:

```
#INCLUDE "initcode.inc"  -- contains definition of PROCESS.SIZE
PROC  program ([PROCESS.SIZE]ProcessData)
```

Note that the program name can be any valid occam identifier.

The single parameter is an array containing information about the process. This information defines the memory map and contains information about extra parameters to the process.

The array also contains a field which is the address of a further array which contains information about the processor.

A detailed explanation of this interface is given in [4].

References

[1] David May and Roger Shepherd.
The Transputer Implementation of occam.
INMOS Technical Note 21.

[2] INMOS Ltd.
Transputer Instruction Set (A compiler writers guide).

[3] Malcolm Boffey.
A Software Implementation of Virtual Links.

[4] Antony King.
Configured Executable Interface.
SW-0029.