

# CC1B Libraries Reference Manual

Generated by Doxygen 1.5.3

Thu Sep 13 19:00:03 2007



# Contents

<b>1</b>	<b>CC1B Libraries File Index</b>	<b>1</b>
1.1	CC1B Libraries File List . . . . .	1
<b>2</b>	<b>CC1B Libraries File Documentation</b>	<b>3</b>
2.1	lib/datastructures/queue/objQueue.h File Reference . . . . .	3
2.2	lib/datastructures/stack/objStack.h File Reference . . . . .	9
2.3	lib/sxdevice/SX18.H File Reference . . . . .	14
2.4	lib/sxdevice/SX20.H File Reference . . . . .	17
2.5	lib/sxdevice/SX28.H File Reference . . . . .	20
2.6	lib/sxdevice/SX48.H File Reference . . . . .	23
2.7	lib/sxdevice/SX52.H File Reference . . . . .	26
2.8	lib/system/defines.h File Reference . . . . .	29
2.9	lib/system/memory.h File Reference . . . . .	42
2.10	lib/system/portpin.h File Reference . . . . .	49
2.11	lib/taskswitching/Task.h File Reference . . . . .	59
2.12	lib/text/character/ctype.h File Reference . . . . .	68
2.13	lib/text/conversion/stdlib.h File Reference . . . . .	73
2.14	lib/text/string/cstring.h File Reference . . . . .	80
2.15	lib/text/string/dstring.h File Reference . . . . .	84
2.16	lib/text/string/string.h File Reference . . . . .	89
2.17	lib/virtualperipheral/uart/vpUart.h File Reference . . . . .	93
2.18	lib/virtualperipheral/vplib.h File Reference . . . . .	99



# Chapter 1

## CC1B Libraries File Index

### 1.1 CC1B Libraries File List

Here is a list of all files with brief descriptions:

<a href="#">lib/datastructures/queue/objQueue.h</a> (Library for queue object ) . . . . .	3
<a href="#">lib/datastructures/stack/objStack.h</a> (Library for stack object ) . . . . .	9
<a href="#">lib/sxdevice/SX18.H</a> (Definition file for SX18 chip ) . . . . .	14
<a href="#">lib/sxdevice/SX20.H</a> (Definition file for SX20 chip ) . . . . .	17
<a href="#">lib/sxdevice/SX28.H</a> (Definition file for SX28 chip ) . . . . .	20
<a href="#">lib/sxdevice/SX48.H</a> (Definition file for SX48 chip ) . . . . .	23
<a href="#">lib/sxdevice/SX52.H</a> (Definition file for SX52 chip ) . . . . .	26
<a href="#">lib/system/defines.h</a> (Definitions for (hardware) resources ) . . . . .	29
<a href="#">lib/system/memory.h</a> (Library for memory access ) . . . . .	42
<a href="#">lib/system/portpin.h</a> (Library for port and pin access ) . . . . .	49
<a href="#">lib/taskswitching/Task.h</a> (Library for taskswitching support ) . . . . .	59
<a href="#">lib/text/character/ctype.h</a> (Library for character functions ) . . . . .	68
<a href="#">lib/text/conversion/stdlib.h</a> (Library for string conversion support ) . . . . .	73
<a href="#">lib/text/string/cstring.h</a> (Library for constant string support ) . . . . .	80
<a href="#">lib/text/string/dstring.h</a> (Library for constant string support ) . . . . .	84
<a href="#">lib/text/string/string.h</a> (Library for string support ) . . . . .	89
<a href="#">lib/virtualperipheral/vplib.h</a> (Library for virtual peripheral support ) . . . . .	99
<a href="#">lib/virtualperipheral/uart/vpUart.h</a> (Library for virtual peripheral vpUart ) . . . . .	93



# Chapter 2

## CC1B Libraries File Documentation

### 2.1 lib/datastructures/queue/objQueue.h File Reference

Library for queue object.

```
#include <system/memory.h>
```

#### Defines

- #define `objQueue_enqueue`(queue, value)  
*Enqueue byte into queue.*
- #define `objQueue_init`(queue, size)  
*Initialize a queue.*

#### Functions

- char `objQueue_capacity` (char W)  
*Get queue capacity.*
- void `objQueue_clear` (char W)  
*Clear queue.*
- char `objQueue_dequeue` (char W)  
*Dequeue byte from queue.*
- char `objQueue_free` (char W)  
*Get queue free space.*
- bit `objQueue_isEmpty` (char W)  
*Check if queue is empty.*
- bit `objQueue_isFull` (char W)  
*Check if queue is full.*

- char [objQueue\\_length](#) (char W)  
*Get queue length.*
- char [objQueue\\_size](#) (char W)  
*Get queue size.*

### 2.1.1 Detailed Description

This library provides a queue object that can store up to 14 bytes. The queue must reside in a single rambank. The functions are written so that they can be used in any call tree (interrupt, mainlevel and tasklevel) by using respectively `_IsrTemp`, `_MainTemp` or `_TaskTemp` as the ram location.

Any character array that resides in a single rambank can be initialized to be a queue.

```
char q[12];
_MainTemp = 8; \\size of the queue must be stored in a global ram location
objQueue_init(q, _MainTemp);
```

Bytes are stored into the queue by using function [objQueue\\_enqueue\(\)](#).

This function requires the value to be enqueued to be stored in a global ram location.

```
if (!objQueue_isFull(q)) {
    _MainTemp = value; //value to enqueue must be in a global ram location
    objQueue_enqueue(q, _MainTemp);
}
```

Bytes are retrieved from the queue by using function [objQueue\\_dequeue\(\)](#).

The retrieved byte is returned in W if the queue is not empty.

```
if (!objQueue_isEmpty(q)) {
    value = objQueue_dequeue(q);
}
```

Definition in file [objQueue.h](#).

### 2.1.2 Define Documentation

#### 2.1.2.1 #define objQueue\_enqueue(queue, value)

The function [objQueue\\_enqueue\(\)](#) stores a byte into a queue. If the queue is already full, the byte is not stored. This is to protect the queue integrity. To know that a byte is stored, this function should be used together with the [objQueue\\_isFull\(\)](#) function.

How to use:

```
if (!objQueue_isFull(q)) {
    _MainTemp = value; //value to enqueue must be in a global ram location
    objQueue_enqueue(q, _MainTemp);
}
```



**Parameters:**

*queue* Queue address.

*value* Byte to enqueue. The value MUST have been stored in a global ram location (0x00-0x0F).

Definition at line 344 of file objQueue.h.

**2.1.2.2 #define objQueue\_init(queue, size)**

The function `objQueue_init()` initializes a character array as a queue object. The character array must reside in a single rambank. The overhead for a queue object is 2 bytes. So for a queue that can hold 6 bytes, a size of 8 bytes must be specified.

How to use:

```
char q[12];
_MainTemp = 12; //size of queue must be in a global ram location
objQueue_init(q, _MainTemp);
```

**Parameters:**

*queue* Queue address

*size* Size of the queue. The size MUST have been stored in a global ram location (0x00-0x0F).

Valid range for size is 3 to 16 (depends on queue address). The queue cannot cross a rambank boundary.

Definition at line 71 of file objQueue.h.

**2.1.3 Function Documentation****2.1.3.1 char objQueue\_capacity(char W)**

The function `objQueue_capacity()` calculates the maximum number of bytes that can be stored into the queue. This capacity equals the queue size minus the overhead (2 bytes).

How to use:

```
capacity = objQueue_capacity(q);
```

**Parameters:**

*W* Queue address.

**Returns:**

Capacity of queue (maximum 14).

Definition at line 210 of file objQueue.h.

### 2.1.3.2 void objQueue\_clear (char W)

The function `ObjQueue_clear()` clears a queue, in effect removing any data it may hold.

This is established by resetting its internal pointers.

How to use:

```
objQueue_clear(q);
```

#### Parameters:

*W* Queue address.

Definition at line 157 of file `objQueue.h`.

### 2.1.3.3 char objQueue\_dequeue (char W)

The function `objQueue_dequeue()` retrieves a byte from a queue. If the queue is already empty, a void value is returned. This is to protect the queue integrity. To know that a byte is retrieved, this function should be used together with the `objQueue_isEmpty()` function.

How to use:

```
if (!objQueue_isEmpty(q)) {  
    value = objQueue_dequeue(q);  
}
```

#### Parameters:

*W* Queue address.

#### Returns:

Byte retrieved from queue (if queue is not empty).

Definition at line 369 of file `objQueue.h`.

### 2.1.3.4 char objQueue\_free (char W)

The function `objQueue_free()` calculates how many additional bytes can be stored in a queue before it is full. This free space equals capacity minus length.

How to use:

```
free = objQueue_free(q);
```

#### Parameters:

*W* Queue address.

#### Returns:

Number of additional bytes the queue can hold (0 to queue capacity).

Definition at line 265 of file `objQueue.h`.

### 2.1.3.5 bit objQueue\_isEmpty (char W)

The function `objQueue_isEmpty()` checks whether a queue holds any data.

Use this function together with `objQueue_dequeue()` to be sure a queue holds any data.

How to use:

```
if (!objQueue_isEmpty(q)) {
    value = objQueue_dequeue(q);
}
```

#### Parameters:

*W* Queue address.

#### Returns:

True (1) if queue empty, false (0) if not empty.

Definition at line 103 of file `objQueue.h`.

### 2.1.3.6 bit objQueue\_isFull (char W)

The function `objQueue_isFull()` checks whether a queue can hold another byte.

Use this function together with `objQueue_enqueue()` to be sure a queue holds more data.

How to use:

```
if (!objQueue_isFull(q)) {
    _MainTemp = value; //value to enqueue must be in a global ram location
    objQueue_enqueue(q, _MainTemp);
}
```

#### Parameters:

*W* Queue address.

#### Returns:

True (1) if queue full, false (0) if not full.

Definition at line 132 of file `objQueue.h`.

### 2.1.3.7 char objQueue\_length (char W)

The function `objQueue_length()` calculates the number of bytes stored in the queue.

This can be up to the capacity of the queue.

How to use:

```
length = objQueue_length(q);
```

#### Parameters:

*W* Queue address.

**Returns:**

Number of stored bytes (0 to queue capacity).

Definition at line 184 of file objQueue.h.

**2.1.3.8 char objQueue\_size (char *W*)**

The function `objQueue_size()` calculates the size of a queue.

This is the size as specified during the last initialization of the queue.

How to use:

```
size = objQueue_size(q);
```

**Parameters:**

*W* Queue address.

**Returns:**

Size of queue (maximum 16).

Definition at line 236 of file objQueue.h.

## 2.2 lib/datastructures/stack/objStack.h File Reference

Library for stack object.

```
#include <system/memory.h>
```

### Defines

- #define `objStack_init`(stack, size)  
*Initialize a stack.*
- #define `objStack_push`(stack, value)  
*Push byte onto stack.*

### Functions

- char `objStack_capacity` (char W)  
*Get stack capacity.*
- void `objStack_clear` (char W)  
*Clear stack.*
- char `objStack_free` (char W)  
*Get stack free space.*
- bit `objStack_isEmpty` (char W)  
*Check if stack is empty.*
- bit `objStack_isFull` (char W)  
*Check if stack is full.*
- char `objStack_length` (char W)  
*Get stack length.*
- char `objStack_pop` (char W)  
*Pop byte from stack.*
- char `objStack_size` (char W)  
*Get stack size.*

### 2.2.1 Detailed Description

This library provides a stack object that can store up to 15 bytes. The stack must reside in a single rambank. The functions are written so that they can be used in any call tree (interrupt, mainlevel and tasklevel) by using respectively `_IsrTemp`, `_MainTemp` or `_TaskTemp` as the ram location. Any character array that resides in a single rambank can be initialized to be a stack.

```
char s[12];
_MainTemp = 8; \\size of the stack must be stored in a global ram location
objStack_init(s,_MainTemp);
```

Bytes are pushed onto the stack by using function [objStack\\_push\(\)](#).

This function requires the value to be pushed to be stored in a global ram location.

```
if (!objStack_isFull(s)) {
    _MainTemp = value; //value to push must be in a global ram location
    objStack_push(s,_MainTemp);
}
```

Bytes are popped from the stack by using function [objStack\\_pop\(\)](#).

The popped byte is returned in W if the stack is not empty.

```
if (!objStack_isEmpty(s)) {
    value = objStack_pop(s);
}
```

Definition in file [objStack.h](#).

## 2.2.2 Define Documentation

### 2.2.2.1 #define objStack\_init(stack, size)

The function [objStack\\_init\(\)](#) initializes a character array as a stack object. The character array must reside in a single rambank. The overhead for a stack object is 1 byte. So for a stack that can hold 6 bytes, a size of 7 bytes must be specified.

How to use:

```
char s[12];
_MainTemp = 12; //size of stack must be in a global ram location
objStack_init(s,_MainTemp);
```

#### Parameters:

**stack** Stack address

**size** Size of the stack. The size MUST have been stored in a global ram location (0x00-0x0F).

Valid range for size is 2 to 16 (depends on stack address). The stack cannot cross a rambank boundary.

Definition at line 69 of file [objStack.h](#).

### 2.2.2.2 #define objStack\_push(stack, value)

The function [objStack\\_push\(\)](#) pushes a byte onto a stack. If the stack is already full, the byte is not stored. This is to protect the stack integrity. To know that a byte is pushed, this function should be used together with the [objStack\\_isFull\(\)](#) function.

How to use:

```
if (!objStack_isFull(s)) {
    _MainTemp = value; //value to push must be in a global ram location
    objStack_push(s, _MainTemp);
}
```

**Parameters:**

*stack* Stack address.

*value* Byte to push. The value MUST have been stored in a global ram location (0x00-0x0F).

Definition at line 328 of file objStack.h.

## 2.2.3 Function Documentation

### 2.2.3.1 char objStack\_capacity (char W)

The function `objStack_capacity()` calculates the maximum number of bytes that can be pushed onto the stack. This capacity equals the stack size minus the overhead (1 byte).

How to use:

```
capacity = objStack_capacity(s);
```

**Parameters:**

*W* Stack address.

**Returns:**

Capacity of stack (maximum 15).

Definition at line 203 of file objStack.h.

### 2.2.3.2 void objStack\_clear (char W)

The function `ObjStack_clear()` clears a stack, in effect removing any data it may hold.

This is established by resetting its internal pointers.

How to use:

```
objStack_clear(s);
```

**Parameters:**

*W* Stack address.

Definition at line 152 of file objStack.h.

### 2.2.3.3 char objStack\_free (char W)

The function `objStack_free()` calculates how many additional bytes can be pushed onto a stack before it is full. This free space equals capacity minus length.

How to use:

```
free = objStack_free(s);
```

**Parameters:**

*W* Stack address.

**Returns:**

Number of additional bytes the stack can hold (0 to stack capacity).

Definition at line 258 of file objStack.h.

**2.2.3.4 bit objStack\_isEmpty (char *W*)**

The function `objStack_isEmpty()` checks whether a stack holds any data.

Use this function together with `objStack_pop()` to be sure a stack holds any data.

How to use:

```
if (!objStack_isEmpty(s)) {  
    value = objStack_pop(s);  
}
```

**Parameters:**

*W* Stack address.

**Returns:**

True (1) if stack empty, false (0) if not empty.

Definition at line 98 of file objStack.h.

**2.2.3.5 bit objStack\_isFull (char *W*)**

The function `objStack_isFull()` checks whether a stack can hold another byte.

Use this function together with `objStack_push()` to be sure a stack holds more data.

How to use:

```
if (!objStack_isFull(s)) {  
    _MainTemp = value; //value to push must be in a global ram location  
    objStack_push(s, _MainTemp);  
}
```

**Parameters:**

*W* Stack address.

**Returns:**

True (1) if stack full, false (0) if not full.

Definition at line 127 of file objStack.h.



### 2.2.3.6 char objStack\_length (char *W*)

The function [objStack\\_length\(\)](#) calculates the number of bytes stored in the stack.

This can be up to the capacity of the stack.

How to use:

```
length = objStack_length(s);
```

**Parameters:**

*W* Stack address.

**Returns:**

Number of stored bytes (0 to stack capacity).

Definition at line 177 of file objStack.h.

### 2.2.3.7 char objStack\_pop (char *W*)

The function [objStack\\_pop\(\)](#) pops a byte from a stack. If the stack is already empty, a void value is returned. This is to protect the stack integrity. To know that a byte is popped, this function should be used together with the [objStack\\_isEmpty\(\)](#) function.

How to use:

```
if (!objStack_isEmpty(s)) {  
    value = objStack_pop(s);  
}
```

**Parameters:**

*W* Stack address.

**Returns:**

Byte popped from stack (if stack is not empty).

Definition at line 353 of file objStack.h.

### 2.2.3.8 char objStack\_size (char *W*)

The function [objStack\\_size\(\)](#) calculates the size of a stack.

This is the size as specified during the last initialization of the stack.

How to use:

```
size = objStack_size(s);
```

**Parameters:**

*W* Stack address.

**Returns:**

Size of stack (maximum 16).

Definition at line 229 of file objStack.h.

## 2.3 lib/sxdevice/SX18.H File Reference

Definition file for SX18 chip.

### Defines

- #define `_CHIP_CODESIZE_` 0x0800
- #define `_CHIP_SX18_`
- #define `membank0` rambank -
- #define `membank1` rambank 0
- #define `membank10` rambank -1
- #define `membank11` rambank 5
- #define `membank12` rambank -1
- #define `membank13` rambank 6
- #define `membank14` rambank -1
- #define `membank15` rambank 7
- #define `membank16` rambank -1
- #define `membank2` rambank -1
- #define `membank3` rambank 1
- #define `membank4` rambank -1
- #define `membank5` rambank 2
- #define `membank6` rambank -1
- #define `membank7` rambank 3
- #define `membank8` rambank -1
- #define `membank9` rambank 4

### 2.3.1 Detailed Description

This file must be the first file to include in a CC1B application for the SX18 chip.

It defines required settings for the CC1B compiler and defines constants for memory banks.

The following memory banks are available for the SX18 chip:

- `membank0` global ram 0x07-0x0F
- `membank1` banked ram 0x10-0x1F
- `membank3` banked ram 0x30-0x3F
- `membank5` banked ram 0x50-0x5F
- `membank7` banked ram 0x70-0x7F
- `membank9` banked ram 0x90-0x9F
- `membank11` banked ram 0xB0-0xBF
- `membank13` banked ram 0xD0-0xDF
- `membank15` banked ram 0xF0-0xFF

Use these constants in #pragma statements to set the current rambank

eg. #pragma `membank1`

Definition in file [SX18.H](#).

## 2.3.2 Define Documentation

### 2.3.2.1 #define \_CHIP\_CODESIZE\_ 0x0800

Definition at line 31 of file SX18.H.

### 2.3.2.2 #define \_CHIP\_SX18\_

Definition at line 30 of file SX18.H.

### 2.3.2.3 #define membank0 rambank -

Definition at line 32 of file SX18.H.

### 2.3.2.4 #define membank1 rambank 0

Definition at line 33 of file SX18.H.

### 2.3.2.5 #define membank10 rambank -1

Definition at line 42 of file SX18.H.

### 2.3.2.6 #define membank11 rambank 5

Definition at line 43 of file SX18.H.

### 2.3.2.7 #define membank12 rambank -1

Definition at line 44 of file SX18.H.

### 2.3.2.8 #define membank13 rambank 6

Definition at line 45 of file SX18.H.

### 2.3.2.9 #define membank14 rambank -1

Definition at line 46 of file SX18.H.

### 2.3.2.10 #define membank15 rambank 7

Definition at line 47 of file SX18.H.

### 2.3.2.11 #define membank16 rambank -1

Definition at line 48 of file SX18.H.

**2.3.2.12 #define membank2 rambank -1**

Definition at line 34 of file SX18.H.

**2.3.2.13 #define membank3 rambank 1**

Definition at line 35 of file SX18.H.

**2.3.2.14 #define membank4 rambank -1**

Definition at line 36 of file SX18.H.

**2.3.2.15 #define membank5 rambank 2**

Definition at line 37 of file SX18.H.

**2.3.2.16 #define membank6 rambank -1**

Definition at line 38 of file SX18.H.

**2.3.2.17 #define membank7 rambank 3**

Definition at line 39 of file SX18.H.

**2.3.2.18 #define membank8 rambank -1**

Definition at line 40 of file SX18.H.

**2.3.2.19 #define membank9 rambank 4**

Definition at line 41 of file SX18.H.

## 2.4 lib/sxdevice/SX20.H File Reference

Definition file for SX20 chip.

### Defines

- #define `_CHIP_CODESIZE_` 0x0800
- #define `_CHIP_SX20_`
- #define `membank0` rambank -
- #define `membank1` rambank 0
- #define `membank10` rambank -1
- #define `membank11` rambank 5
- #define `membank12` rambank -1
- #define `membank13` rambank 6
- #define `membank14` rambank -1
- #define `membank15` rambank 7
- #define `membank16` rambank -1
- #define `membank2` rambank -1
- #define `membank3` rambank 1
- #define `membank4` rambank -1
- #define `membank5` rambank 2
- #define `membank6` rambank -1
- #define `membank7` rambank 3
- #define `membank8` rambank -1
- #define `membank9` rambank 4

### 2.4.1 Detailed Description

This file must be the first file to include in a CC1B application for the SX20 chip.

It defines required settings for the CC1B compiler and defines constants for memory banks.

The following memory banks are available for the SX20 chip:

- `membank0` global ram 0x07-0x0F
- `membank1` banked ram 0x10-0x1F
- `membank3` banked ram 0x30-0x3F
- `membank5` banked ram 0x50-0x5F
- `membank7` banked ram 0x70-0x7F
- `membank9` banked ram 0x90-0x9F
- `membank11` banked ram 0xB0-0xBF
- `membank13` banked ram 0xD0-0xDF
- `membank15` banked ram 0xF0-0xFF

Use these constants in #pragma statements to set the current rambank

eg. #pragma membank 1

Definition in file [SX20.H](#).

## 2.4.2 Define Documentation

### 2.4.2.1 **#define \_CHIP\_CODESIZE\_ 0x0800**

Definition at line 31 of file SX20.H.

### 2.4.2.2 **#define \_CHIP\_SX20\_**

Definition at line 30 of file SX20.H.

### 2.4.2.3 **#define membank0 rambank -**

Definition at line 32 of file SX20.H.

### 2.4.2.4 **#define membank1 rambank 0**

Definition at line 33 of file SX20.H.

### 2.4.2.5 **#define membank10 rambank -1**

Definition at line 42 of file SX20.H.

### 2.4.2.6 **#define membank11 rambank 5**

Definition at line 43 of file SX20.H.

### 2.4.2.7 **#define membank12 rambank -1**

Definition at line 44 of file SX20.H.

### 2.4.2.8 **#define membank13 rambank 6**

Definition at line 45 of file SX20.H.

### 2.4.2.9 **#define membank14 rambank -1**

Definition at line 46 of file SX20.H.

### 2.4.2.10 **#define membank15 rambank 7**

Definition at line 47 of file SX20.H.

### 2.4.2.11 **#define membank16 rambank -1**

Definition at line 48 of file SX20.H.

**2.4.2.12 #define membank2 rambank -1**

Definition at line 34 of file SX20.H.

**2.4.2.13 #define membank3 rambank 1**

Definition at line 35 of file SX20.H.

**2.4.2.14 #define membank4 rambank -1**

Definition at line 36 of file SX20.H.

**2.4.2.15 #define membank5 rambank 2**

Definition at line 37 of file SX20.H.

**2.4.2.16 #define membank6 rambank -1**

Definition at line 38 of file SX20.H.

**2.4.2.17 #define membank7 rambank 3**

Definition at line 39 of file SX20.H.

**2.4.2.18 #define membank8 rambank -1**

Definition at line 40 of file SX20.H.

**2.4.2.19 #define membank9 rambank 4**

Definition at line 41 of file SX20.H.

## 2.5 lib/sxdevice/SX28.H File Reference

Definition file for SX28 chip.

### Defines

- #define `_CHIP_CODESIZE_` 0x0800
- #define `_CHIP_SX28_`
- #define `membank0` rambank -
- #define `membank1` rambank 0
- #define `membank10` rambank -1
- #define `membank11` rambank 5
- #define `membank12` rambank -1
- #define `membank13` rambank 6
- #define `membank14` rambank -1
- #define `membank15` rambank 7
- #define `membank16` rambank -1
- #define `membank2` rambank -1
- #define `membank3` rambank 1
- #define `membank4` rambank -1
- #define `membank5` rambank 2
- #define `membank6` rambank -1
- #define `membank7` rambank 3
- #define `membank8` rambank -1
- #define `membank9` rambank 4

### 2.5.1 Detailed Description

This file must be the first file to include in a CC1B application for the SX28 chip.

It defines required settings for the CC1B compiler and defines constants for memory banks.

The following memory banks are available for the SX28 chip:

- `membank0` global ram 0x08-0x0F
- `membank1` banked ram 0x10-0x1F
- `membank3` banked ram 0x30-0x3F
- `membank5` banked ram 0x50-0x5F
- `membank7` banked ram 0x70-0x7F
- `membank9` banked ram 0x90-0x9F
- `membank11` banked ram 0xB0-0xBF
- `membank13` banked ram 0xD0-0xDF
- `membank15` banked ram 0xF0-0xFF

Use these constants in #pragma statements to set the current rambank

eg. #pragma membank 1

Definition in file [SX28.H](#).



## 2.5.2 Define Documentation

### 2.5.2.1 **#define \_CHIP\_CODESIZE\_ 0x0800**

Definition at line 31 of file SX28.H.

### 2.5.2.2 **#define \_CHIP\_SX28\_**

Definition at line 30 of file SX28.H.

### 2.5.2.3 **#define membank0 rambank -**

Definition at line 32 of file SX28.H.

### 2.5.2.4 **#define membank1 rambank 0**

Definition at line 33 of file SX28.H.

### 2.5.2.5 **#define membank10 rambank -1**

Definition at line 42 of file SX28.H.

### 2.5.2.6 **#define membank11 rambank 5**

Definition at line 43 of file SX28.H.

### 2.5.2.7 **#define membank12 rambank -1**

Definition at line 44 of file SX28.H.

### 2.5.2.8 **#define membank13 rambank 6**

Definition at line 45 of file SX28.H.

### 2.5.2.9 **#define membank14 rambank -1**

Definition at line 46 of file SX28.H.

### 2.5.2.10 **#define membank15 rambank 7**

Definition at line 47 of file SX28.H.

### 2.5.2.11 **#define membank16 rambank -1**

Definition at line 48 of file SX28.H.

**2.5.2.12 #define membank2 rambank -1**

Definition at line 34 of file SX28.H.

**2.5.2.13 #define membank3 rambank 1**

Definition at line 35 of file SX28.H.

**2.5.2.14 #define membank4 rambank -1**

Definition at line 36 of file SX28.H.

**2.5.2.15 #define membank5 rambank 2**

Definition at line 37 of file SX28.H.

**2.5.2.16 #define membank6 rambank -1**

Definition at line 38 of file SX28.H.

**2.5.2.17 #define membank7 rambank 3**

Definition at line 39 of file SX28.H.

**2.5.2.18 #define membank8 rambank -1**

Definition at line 40 of file SX28.H.

**2.5.2.19 #define membank9 rambank 4**

Definition at line 41 of file SX28.H.

## 2.6 lib/sxdevice/SX48.H File Reference

Definition file for SX48 chip.

### Defines

- #define `_CHIP_CODESIZE_` 0x1000
- #define `_CHIP_SX48_`
- #define `membank0` rambank -
- #define `membank1` rambank 1
- #define `membank10` rambank 10
- #define `membank11` rambank 11
- #define `membank12` rambank 12
- #define `membank13` rambank 13
- #define `membank14` rambank 14
- #define `membank15` rambank 15
- #define `membank16` rambank 0
- #define `membank2` rambank 2
- #define `membank3` rambank 3
- #define `membank4` rambank 4
- #define `membank5` rambank 5
- #define `membank6` rambank 6
- #define `membank7` rambank 7
- #define `membank8` rambank 8
- #define `membank9` rambank 9

### 2.6.1 Detailed Description

This file must be the first file to include in a CC1B application for the SX48 chip.

It defines required settings for the CC1B compiler and defines constants for memory banks.

The following memory banks are available for the SX48 chip:

- `membank0` global ram 0x0A-0x0F
- `membank1` banked ram 0x10-0x1F
- `membank2` banked ram 0x20-0x2F
- `membank3` banked ram 0x30-0x3F
- `membank4` banked ram 0x40-0x4F
- `membank5` banked ram 0x50-0x5F
- `membank6` banked ram 0x60-0x6F
- `membank7` banked ram 0x70-0x7F
- `membank8` banked ram 0x80-0x8F
- `membank9` banked ram 0x90-0x9F
- `membank10` banked ram 0xA0-0xAF

- membank11 banked ram 0xB0-0xBF
- membank12 banked ram 0xC0-0xCF
- membank13 banked ram 0xD0-0xDF
- membank14 banked ram 0xE0-0xEF
- membank15 banked ram 0xF0-0xFF
- membank16 banked ram 0x00-0x0F (semi-direct addressing)

Use these constants in #pragma statements to set the current rambank

eg. #pragma membank1

Definition in file [SX48.H](#).

## 2.6.2 Define Documentation

### 2.6.2.1 #define \_CHIP\_CODESIZE\_ 0x1000

Definition at line 39 of file SX48.H.

### 2.6.2.2 #define \_CHIP\_SX48\_

Definition at line 38 of file SX48.H.

### 2.6.2.3 #define membank0 rambank -

Definition at line 40 of file SX48.H.

### 2.6.2.4 #define membank1 rambank 1

Definition at line 41 of file SX48.H.

### 2.6.2.5 #define membank10 rambank 10

Definition at line 50 of file SX48.H.

### 2.6.2.6 #define membank11 rambank 11

Definition at line 51 of file SX48.H.

### 2.6.2.7 #define membank12 rambank 12

Definition at line 52 of file SX48.H.

### 2.6.2.8 #define membank13 rambank 13

Definition at line 53 of file SX48.H.

**2.6.2.9 #define membank14 rambank 14**

Definition at line 54 of file SX48.H.

**2.6.2.10 #define membank15 rambank 15**

Definition at line 55 of file SX48.H.

**2.6.2.11 #define membank16 rambank 0**

Definition at line 56 of file SX48.H.

**2.6.2.12 #define membank2 rambank 2**

Definition at line 42 of file SX48.H.

**2.6.2.13 #define membank3 rambank 3**

Definition at line 43 of file SX48.H.

**2.6.2.14 #define membank4 rambank 4**

Definition at line 44 of file SX48.H.

**2.6.2.15 #define membank5 rambank 5**

Definition at line 45 of file SX48.H.

**2.6.2.16 #define membank6 rambank 6**

Definition at line 46 of file SX48.H.

**2.6.2.17 #define membank7 rambank 7**

Definition at line 47 of file SX48.H.

**2.6.2.18 #define membank8 rambank 8**

Definition at line 48 of file SX48.H.

**2.6.2.19 #define membank9 rambank 9**

Definition at line 49 of file SX48.H.

## 2.7 lib/sxdevice/SX52.H File Reference

Definition file for SX52 chip.

### Defines

- #define `_CHIP_CODESIZE_` 0x1000
- #define `_CHIP_SX52_`
- #define `membank0` rambank -
- #define `membank1` rambank 1
- #define `membank10` rambank 10
- #define `membank11` rambank 11
- #define `membank12` rambank 12
- #define `membank13` rambank 13
- #define `membank14` rambank 14
- #define `membank15` rambank 15
- #define `membank16` rambank 0
- #define `membank2` rambank 2
- #define `membank3` rambank 3
- #define `membank4` rambank 4
- #define `membank5` rambank 5
- #define `membank6` rambank 6
- #define `membank7` rambank 7
- #define `membank8` rambank 8
- #define `membank9` rambank 9

### 2.7.1 Detailed Description

This file must be the first file to include in a CC1B application for the SX52 chip.

It defines required settings for the CC1B compiler and defines constants for memory banks.

The following memory banks are available for the SX52 chip:

- `membank0` global ram 0x0A-0x0F
- `membank1` banked ram 0x10-0x1F
- `membank2` banked ram 0x20-0x2F
- `membank3` banked ram 0x30-0x3F
- `membank4` banked ram 0x40-0x4F
- `membank5` banked ram 0x50-0x5F
- `membank6` banked ram 0x60-0x6F
- `membank7` banked ram 0x70-0x7F
- `membank8` banked ram 0x80-0x8F
- `membank9` banked ram 0x90-0x9F
- `membank10` banked ram 0xA0-0xAF

- membank11 banked ram 0xB0-0xBF
- membank12 banked ram 0xC0-0xCF
- membank13 banked ram 0xD0-0xDF
- membank14 banked ram 0xE0-0xEF
- membank15 banked ram 0xF0-0xFF
- membank16 banked ram 0x00-0x0F (semi-direct addressing)

Use these constants in #pragma statements to set the current rambank

eg. #pragma membank1

Definition in file [SX52.H](#).

## 2.7.2 Define Documentation

### 2.7.2.1 #define \_CHIP\_CODESIZE\_ 0x1000

Definition at line 39 of file SX52.H.

### 2.7.2.2 #define \_CHIP\_SX52\_

Definition at line 38 of file SX52.H.

### 2.7.2.3 #define membank0 rambank -

Definition at line 40 of file SX52.H.

### 2.7.2.4 #define membank1 rambank 1

Definition at line 41 of file SX52.H.

### 2.7.2.5 #define membank10 rambank 10

Definition at line 50 of file SX52.H.

### 2.7.2.6 #define membank11 rambank 11

Definition at line 51 of file SX52.H.

### 2.7.2.7 #define membank12 rambank 12

Definition at line 52 of file SX52.H.

### 2.7.2.8 #define membank13 rambank 13

Definition at line 53 of file SX52.H.

**2.7.2.9 #define membank14 rambank 14**

Definition at line 54 of file SX52.H.

**2.7.2.10 #define membank15 rambank 15**

Definition at line 55 of file SX52.H.

**2.7.2.11 #define membank16 rambank 0**

Definition at line 56 of file SX52.H.

**2.7.2.12 #define membank2 rambank 2**

Definition at line 42 of file SX52.H.

**2.7.2.13 #define membank3 rambank 3**

Definition at line 43 of file SX52.H.

**2.7.2.14 #define membank4 rambank 4**

Definition at line 44 of file SX52.H.

**2.7.2.15 #define membank5 rambank 5**

Definition at line 45 of file SX52.H.

**2.7.2.16 #define membank6 rambank 6**

Definition at line 46 of file SX52.H.

**2.7.2.17 #define membank7 rambank 7**

Definition at line 47 of file SX52.H.

**2.7.2.18 #define membank8 rambank 8**

Definition at line 48 of file SX52.H.

**2.7.2.19 #define membank9 rambank 9**

Definition at line 49 of file SX52.H.



## 2.8 lib/system/defines.h File Reference

Definitions for (hardware) resources.

### Defines

- #define `Baud_divide`(baud, ratio)  $((2 * \text{uartfs} / \text{baud} / \text{ratio}) + 1) / 2$
- #define `E71` 0b00001110
- #define `E72` 0b00001111
- #define `E81` 0b00001011
- #define `HIGHINPUT_HIGHINPUT` 0b01110111
- #define `HIGHINPUT_HIGHOUTPUT` 0b01110101
- #define `HIGHINPUT_LEAVE` 0b01110000
- #define `HIGHINPUT_LOWINPUT` 0b01110110
- #define `HIGHINPUT_LOWOUTPUT` 0b01110100
- #define `HIGHOUTPUT_HIGHINPUT` 0b01010111
- #define `HIGHOUTPUT_HIGHOUTPUT` 0b01010101
- #define `HIGHOUTPUT_LEAVE` 0b01010000
- #define `HIGHOUTPUT_LOWINPUT` 0b01010110
- #define `HIGHOUTPUT_LOWOUTPUT` 0b01010100
- #define `INTPERIOD`  $((2 * \text{CPUFREQ} / \text{uartfs}) + 1) / 2$
- #define `LEAVE_HIGHINPUT` 0b00000111
- #define `LEAVE_HIGHOUTPUT` 0b00000101
- #define `LEAVE_LEAVE` 0b00000000
- #define `LEAVE_LOWINPUT` 0b00000110
- #define `LEAVE_LOWOUTPUT` 0b00000100
- #define `LOWINPUT_HIGHINPUT` 0b01100111
- #define `LOWINPUT_HIGHOUTPUT` 0b01100101
- #define `LOWINPUT_LEAVE` 0b01100000
- #define `LOWINPUT_LOWINPUT` 0b01100110
- #define `LOWINPUT_LOWOUTPUT` 0b01100100
- #define `LOWOUTPUT_HIGHOUTPUT` 0b01000101
- #define `LOWOUTPUT_HIGHINPUT` 0b01000111
- #define `LOWOUTPUT_LEAVE` 0b01000000
- #define `LOWOUTPUT_LOWINPUT` 0b01000110
- #define `LOWOUTPUT_LOWOUTPUT` 0b01000100
- #define `MAXBAUD` 115200
- #define `MSCOUNT`  $((2 * 1000 * (\text{CPUFREQ} / 1000000) / \text{INTPERIOD}) + 1) / 2$
- #define `N71` 0b00000101
- #define `N72` 0b00000110
- #define `N81` 0b00000010
- #define `N82` 0b00000011
- #define `O71` 0b00010110
- #define `O72` 0b00010111
- #define `O81` 0b00010011
- #define `PORT_RA` 0x05
- #define `PORT_RB` 0x06
- #define `PORT_RC` 0x07
- #define `PORT_RD` 0x08

- #define [PORT\\_RE](#) 0x09
- #define [PORTPIN\\_INV](#) 0x1000
- #define [PORTPIN\\_OC](#) 0x2000
- #define [PORTPIN\\_RA](#) 0x05FF
- #define [PORTPIN\\_RA0](#) 0x0501
- #define [PORTPIN\\_RA1](#) 0x0502
- #define [PORTPIN\\_RA2](#) 0x0504
- #define [PORTPIN\\_RA3](#) 0x0508
- #define [PORTPIN\\_RA4](#) 0x0510
- #define [PORTPIN\\_RA5](#) 0x0520
- #define [PORTPIN\\_RA6](#) 0x0540
- #define [PORTPIN\\_RA7](#) 0x0580
- #define [PORTPIN\\_RB](#) 0x06FF
- #define [PORTPIN\\_RB0](#) 0x0601
- #define [PORTPIN\\_RB1](#) 0x0602
- #define [PORTPIN\\_RB2](#) 0x0604
- #define [PORTPIN\\_RB3](#) 0x0608
- #define [PORTPIN\\_RB4](#) 0x0610
- #define [PORTPIN\\_RB5](#) 0x0620
- #define [PORTPIN\\_RB6](#) 0x0640
- #define [PORTPIN\\_RB7](#) 0x0680
- #define [PORTPIN\\_RC](#) 0x07FF
- #define [PORTPIN\\_RC0](#) 0x0701
- #define [PORTPIN\\_RC1](#) 0x0702
- #define [PORTPIN\\_RC2](#) 0x0704
- #define [PORTPIN\\_RC3](#) 0x0708
- #define [PORTPIN\\_RC4](#) 0x0710
- #define [PORTPIN\\_RC5](#) 0x0720
- #define [PORTPIN\\_RC6](#) 0x0740
- #define [PORTPIN\\_RC7](#) 0x0780
- #define [PORTPIN\\_RD](#) 0x08FF
- #define [PORTPIN\\_RD0](#) 0x0801
- #define [PORTPIN\\_RD1](#) 0x0802
- #define [PORTPIN\\_RD2](#) 0x0804
- #define [PORTPIN\\_RD3](#) 0x0808
- #define [PORTPIN\\_RD4](#) 0x0810
- #define [PORTPIN\\_RD5](#) 0x0820
- #define [PORTPIN\\_RD6](#) 0x0840
- #define [PORTPIN\\_RD7](#) 0x0880
- #define [PORTPIN\\_RE](#) 0x09FF
- #define [PORTPIN\\_RE0](#) 0x0901
- #define [PORTPIN\\_RE1](#) 0x0902
- #define [PORTPIN\\_RE2](#) 0x0904
- #define [PORTPIN\\_RE3](#) 0x0908
- #define [PORTPIN\\_RE4](#) 0x0910
- #define [PORTPIN\\_RE5](#) 0x0920
- #define [PORTPIN\\_RE6](#) 0x0940
- #define [PORTPIN\\_RE7](#) 0x0980
- #define [PS\\_000](#) 0b00000000
- #define [PS\\_001](#) 0b00000001

- #define `PS_010` 0b00000010
- #define `PS_011` 0b00000011
- #define `PS_100` 0b00000100
- #define `PS_101` 0b00000101
- #define `PS_110` 0b00000110
- #define `PS_111` 0b00000111
- #define `RTCC_FE` 0b00010000
- #define `RTCC_ID` 0b01000000
- #define `RTCC_INC_EXT` 0b00100000
- #define `RTCC_ON` 0b10000000
- #define `RTCC_PS_OFF` 0b00001000
- #define `RTCC_PS_ON` 0b00000000
- #define `uartfs` 230400
- #define `uartfs` MAXBAUD\*2
- #define `USCOUNT` ((2\*250\*(CPUFREQ/1000000)/INTPERIOD)+1)/2
- #define `VP_TIMER16` 1
- #define `VP_UARTRX` 2
- #define `VP_UARTTX` 3
- #define `VP_UNINSTALLED` 0

### 2.8.1 Detailed Description

This file contains definitions for OPTION register, ports and pins, macros for baudrate calculation.

Definition in file [defines.h](#).

### 2.8.2 Define Documentation

#### 2.8.2.1 #define `Baud_divide(baud, ratio)` ((2 \* `uartfs` / `baud` / `ratio`) + 1) / 2

Definition at line 226 of file [defines.h](#).

#### 2.8.2.2 #define `E71` 0b00001110

Definition at line 103 of file [defines.h](#).

#### 2.8.2.3 #define `E72` 0b00001111

Definition at line 104 of file [defines.h](#).

#### 2.8.2.4 #define `E81` 0b00001011

Definition at line 105 of file [defines.h](#).

#### 2.8.2.5 #define `HIGHINPUT_HIGHINPUT` 0b01110111

Definition at line 144 of file [defines.h](#).

**2.8.2.6 #define HIGHINPUT\_HIGHOUTPUT 0b01110101**

Definition at line 142 of file defines.h.

**2.8.2.7 #define HIGHINPUT\_LEAVE 0b01110000**

Definition at line 140 of file defines.h.

**2.8.2.8 #define HIGHINPUT\_LOWINPUT 0b01110110**

Definition at line 143 of file defines.h.

**2.8.2.9 #define HIGHINPUT\_LOWOUTPUT 0b01110100**

Definition at line 141 of file defines.h.

**2.8.2.10 #define HIGHOUTPUT\_HIGHINPUT 0b01010111**

Definition at line 134 of file defines.h.

**2.8.2.11 #define HIGHOUTPUT\_HIGHOUTPUT 0b01010101**

Definition at line 132 of file defines.h.

**2.8.2.12 #define HIGHOUTPUT\_LEAVE 0b01010000**

Definition at line 130 of file defines.h.

**2.8.2.13 #define HIGHOUTPUT\_LOWINPUT 0b01010110**

Definition at line 133 of file defines.h.

**2.8.2.14 #define HIGHOUTPUT\_LOWOUTPUT 0b01010100**

Definition at line 131 of file defines.h.

**2.8.2.15 #define INTPERIOD ((2\*CPUFREQ/uartfs)+1)/2**

Definition at line 193 of file defines.h.

**2.8.2.16 #define LEAVE\_HIGHINPUT 0b00000111**

Definition at line 124 of file defines.h.

**2.8.2.17 #define LEAVE\_HIGHOUTPUT 0b00000101**

Definition at line 122 of file defines.h.

**2.8.2.18 #define LEAVE\_LEAVE 0b00000000**

Definition at line 120 of file defines.h.

**2.8.2.19 #define LEAVE\_LOWINPUT 0b00000110**

Definition at line 123 of file defines.h.

**2.8.2.20 #define LEAVE\_LOWOUTPUT 0b00000100**

Definition at line 121 of file defines.h.

**2.8.2.21 #define LOWINPUT\_HIGHINPUT 0b01100111**

Definition at line 139 of file defines.h.

**2.8.2.22 #define LOWINPUT\_HIGHOUTPUT 0b01100101**

Definition at line 137 of file defines.h.

**2.8.2.23 #define LOWINPUT\_LEAVE 0b01100000**

Definition at line 135 of file defines.h.

**2.8.2.24 #define LOWINPUT\_LOWINPUT 0b01100110**

Definition at line 138 of file defines.h.

**2.8.2.25 #define LOWINPUT\_LOWOUTPUT 0b01100100**

Definition at line 136 of file defines.h.

**2.8.2.26 #define LOWOUTPUT\_HIGHOUTPUT 0b01000101**

Definition at line 127 of file defines.h.

**2.8.2.27 #define LOWOUTPUT\_HIGHINPUT 0b01000111**

Definition at line 129 of file defines.h.

**2.8.2.28 #define LOWOUTPUT\_LEAVE 0b01000000**

Definition at line 125 of file defines.h.

**2.8.2.29 #define LOWOUTPUT\_LOWINPUT 0b01000110**

Definition at line 128 of file defines.h.

**2.8.2.30 #define LOWOUTPUT\_LOWOUTPUT 0b01000100**

Definition at line 126 of file defines.h.

**2.8.2.31 #define MAXBAUD 115200**

Definition at line 150 of file defines.h.

**2.8.2.32 #define MSCOUNT ((2\*1000\*(CPUFREQ/1000000)/INTPERIOD)+1)/2**

Definition at line 196 of file defines.h.

**2.8.2.33 #define N71 0b00000101**

Definition at line 109 of file defines.h.

**2.8.2.34 #define N72 0b00000110**

Definition at line 110 of file defines.h.

**2.8.2.35 #define N81 0b00000010**

Definition at line 111 of file defines.h.

**2.8.2.36 #define N82 0b00000011**

Definition at line 112 of file defines.h.

**2.8.2.37 #define O71 0b00010110**

Definition at line 106 of file defines.h.

**2.8.2.38 #define O72 0b00010111**

Definition at line 107 of file defines.h.

**2.8.2.39 #define O81 0b00010011**

Definition at line 108 of file defines.h.

**2.8.2.40 #define PORT\_RA 0x05**

Definition at line 38 of file defines.h.

**2.8.2.41 #define PORT\_RB 0x06**

Definition at line 52 of file defines.h.

**2.8.2.42 #define PORT\_RC 0x07**

Definition at line 64 of file defines.h.

**2.8.2.43 #define PORT\_RD 0x08**

Definition at line 77 of file defines.h.

**2.8.2.44 #define PORT\_RE 0x09**

Definition at line 87 of file defines.h.

**2.8.2.45 #define PORTPIN\_INV 0x1000**

Definition at line 99 of file defines.h.

**2.8.2.46 #define PORTPIN\_OC 0x2000**

Definition at line 100 of file defines.h.

**2.8.2.47 #define PORTPIN\_RA 0x05FF**

Definition at line 39 of file defines.h.

**2.8.2.48 #define PORTPIN\_RA0 0x0501**

Definition at line 40 of file defines.h.

**2.8.2.49 #define PORTPIN\_RA1 0x0502**

Definition at line 41 of file defines.h.

**2.8.2.50 #define PORTPIN\_RA2 0x0504**

Definition at line 42 of file defines.h.

**2.8.2.51 #define PORTPIN\_RA3 0x0508**

Definition at line 43 of file defines.h.

**2.8.2.52 #define PORTPIN\_RA4 0x0510**

Definition at line 46 of file defines.h.

**2.8.2.53 #define PORTPIN\_RA5 0x0520**

Definition at line 47 of file defines.h.

**2.8.2.54 #define PORTPIN\_RA6 0x0540**

Definition at line 48 of file defines.h.

**2.8.2.55 #define PORTPIN\_RA7 0x0580**

Definition at line 49 of file defines.h.

**2.8.2.56 #define PORTPIN\_RB 0x06FF**

Definition at line 53 of file defines.h.

**2.8.2.57 #define PORTPIN\_RB0 0x0601**

Definition at line 54 of file defines.h.

**2.8.2.58 #define PORTPIN\_RB1 0x0602**

Definition at line 55 of file defines.h.

**2.8.2.59 #define PORTPIN\_RB2 0x0604**

Definition at line 56 of file defines.h.

**2.8.2.60 #define PORTPIN\_RB3 0x0608**

Definition at line 57 of file defines.h.



**2.8.2.61 #define PORTPIN\_RB4 0x0610**

Definition at line 58 of file defines.h.

**2.8.2.62 #define PORTPIN\_RB5 0x0620**

Definition at line 59 of file defines.h.

**2.8.2.63 #define PORTPIN\_RB6 0x0640**

Definition at line 60 of file defines.h.

**2.8.2.64 #define PORTPIN\_RB7 0x0680**

Definition at line 61 of file defines.h.

**2.8.2.65 #define PORTPIN\_RC 0x07FF**

Definition at line 65 of file defines.h.

**2.8.2.66 #define PORTPIN\_RC0 0x0701**

Definition at line 66 of file defines.h.

**2.8.2.67 #define PORTPIN\_RC1 0x0702**

Definition at line 67 of file defines.h.

**2.8.2.68 #define PORTPIN\_RC2 0x0704**

Definition at line 68 of file defines.h.

**2.8.2.69 #define PORTPIN\_RC3 0x0708**

Definition at line 69 of file defines.h.

**2.8.2.70 #define PORTPIN\_RC4 0x0710**

Definition at line 70 of file defines.h.

**2.8.2.71 #define PORTPIN\_RC5 0x0720**

Definition at line 71 of file defines.h.

**2.8.2.72 #define PORTPIN\_RC6 0x0740**

Definition at line 72 of file defines.h.

**2.8.2.73 #define PORTPIN\_RC7 0x0780**

Definition at line 73 of file defines.h.

**2.8.2.74 #define PORTPIN\_RD 0x08FF**

Definition at line 78 of file defines.h.

**2.8.2.75 #define PORTPIN\_RD0 0x0801**

Definition at line 79 of file defines.h.

**2.8.2.76 #define PORTPIN\_RD1 0x0802**

Definition at line 80 of file defines.h.

**2.8.2.77 #define PORTPIN\_RD2 0x0804**

Definition at line 81 of file defines.h.

**2.8.2.78 #define PORTPIN\_RD3 0x0808**

Definition at line 82 of file defines.h.

**2.8.2.79 #define PORTPIN\_RD4 0x0810**

Definition at line 83 of file defines.h.

**2.8.2.80 #define PORTPIN\_RD5 0x0820**

Definition at line 84 of file defines.h.

**2.8.2.81 #define PORTPIN\_RD6 0x0840**

Definition at line 85 of file defines.h.

**2.8.2.82 #define PORTPIN\_RD7 0x0880**

Definition at line 86 of file defines.h.

**2.8.2.83 #define PORTPIN\_RE 0x09FF**

Definition at line 88 of file defines.h.

**2.8.2.84 #define PORTPIN\_RE0 0x0901**

Definition at line 89 of file defines.h.

**2.8.2.85 #define PORTPIN\_RE1 0x0902**

Definition at line 90 of file defines.h.

**2.8.2.86 #define PORTPIN\_RE2 0x0904**

Definition at line 91 of file defines.h.

**2.8.2.87 #define PORTPIN\_RE3 0x0908**

Definition at line 92 of file defines.h.

**2.8.2.88 #define PORTPIN\_RE4 0x0910**

Definition at line 93 of file defines.h.

**2.8.2.89 #define PORTPIN\_RE5 0x0920**

Definition at line 94 of file defines.h.

**2.8.2.90 #define PORTPIN\_RE6 0x0940**

Definition at line 95 of file defines.h.

**2.8.2.91 #define PORTPIN\_RE7 0x0980**

Definition at line 96 of file defines.h.

**2.8.2.92 #define PS\_000 0b00000000**

Definition at line 29 of file defines.h.

**2.8.2.93 #define PS\_001 0b00000001**

Definition at line 30 of file defines.h.

**2.8.2.94 #define PS\_010 0b00000010**

Definition at line 31 of file defines.h.

**2.8.2.95 #define PS\_011 0b00000011**

Definition at line 32 of file defines.h.

**2.8.2.96 #define PS\_100 0b00000100**

Definition at line 33 of file defines.h.

**2.8.2.97 #define PS\_101 0b00000101**

Definition at line 34 of file defines.h.

**2.8.2.98 #define PS\_110 0b00000110**

Definition at line 35 of file defines.h.

**2.8.2.99 #define PS\_111 0b00000111**

Definition at line 36 of file defines.h.

**2.8.2.100 #define RTCC\_FE 0b00010000**

Definition at line 25 of file defines.h.

**2.8.2.101 #define RTCC\_ID 0b01000000**

Definition at line 21 of file defines.h.

**2.8.2.102 #define RTCC\_INC\_EXT 0b00100000**

Definition at line 23 of file defines.h.

**2.8.2.103 #define RTCC\_ON 0b10000000**

Definition at line 19 of file defines.h.

**2.8.2.104 #define RTCC\_PS\_OFF 0b00001000**

Definition at line 28 of file defines.h.

**2.8.2.105 #define RTCC\_PS\_ON 0b00000000**

Definition at line 27 of file defines.h.

**2.8.2.106 #define uartfs 230400**

Definition at line 183 of file defines.h.

**2.8.2.107 #define uartfs MAXBAUD\*2**

Definition at line 183 of file defines.h.

**2.8.2.108 #define USCOUNT ((2\*250\*(CPUFREQ/1000000)/INTPERIOD)+1)/2**

Definition at line 198 of file defines.h.

**2.8.2.109 #define VP\_TIMER16 1**

Definition at line 115 of file defines.h.

**2.8.2.110 #define VP\_UARTRX 2**

Definition at line 116 of file defines.h.

**2.8.2.111 #define VP\_UARTTX 3**

Definition at line 117 of file defines.h.

**2.8.2.112 #define VP\_UNINSTALLED 0**

Definition at line 114 of file defines.h.

## 2.9 lib/system/memory.h File Reference

Library for memory access.

### Defines

- #define `LargeArrayRead`(base, index)  
*Read byte from large array.*
- #define `LargeArrayWrite`(base, index, value)  
*Write byte to large array.*
- #define `RomWord`(addrH, addrL)  
*Read word from rom.*

### Functions

- char `_SystemRamAddress` (char W)  
*Calculate physical address for \_SystemRam[] index.*
- char `_SystemRamIndex` (char W)  
*Calculate logical index for \_SystemRam[] from physical address.*
- void `DDR_init` (void)  
*Initialize shadow DDR registers.*
- char `LargeArrayAddress` (char W)  
*Calculate physical address for logical array index.*
- char `LargeArrayIndex` (char W)  
*Calculate logical array index from physical address.*
- void `RomCopy` (char dest, long source, char len)  
*Copy bytes from rom to ram.*

### Variables

- shadowDef char `_SystemRam`[256] `x00`
- char `_IsrBank` `x0C`
- char `_IsrTemp` `x0D`
- char `_MainTemp` `x0E`
- char `DDRACOPY` `xF5`
- char `DDRBCOPY` `xF6`
- char `DDRCCOPY` `xF7`
- char `DDRDCOPY` `xF8`
- char `DDRECOPY` `xF9`

## 2.9.1 Detailed Description

This library provides functions to access both ram and rom in an easy way. These functions hide the fact that ram memory is banked. For optimization reasons, these functions do not perform boundary checks.

Definition in file [memory.h](#).

## 2.9.2 Define Documentation

### 2.9.2.1 #define LargeArrayRead(base, index)

The function [LargeArrayRead\(\)](#) reads a byte from a character array that may cross rambank boundaries. No checks are done for address wrapping around 0xFF.

How to use:

```
LargeArrayRead(base, index);  
value = W;
```

#### Parameters:

*base* Physical ram address (0x10-0xFF) of array element 0.

*index* Logical index into the array.

#### Returns:

Byte read.

Definition at line 398 of file [memory.h](#).

### 2.9.2.2 #define LargeArrayWrite(base, index, value)

The function [LargeArrayWrite\(\)](#) writes a byte to a character array that may cross rambank boundaries. No checks are done for address wrapping around 0xFF.

How to use:

```
_MainTemp = value; //value to write must be stored in global ram location  
LargeArrayWrite(base, index, _MainTemp);
```

#### Parameters:

*base* Physical ram address (0x10-0xFF) of array element 0.

*index* Logical index into the array.

*value* Byte to write.

Definition at line 371 of file [memory.h](#).

### 2.9.2.3 #define RomWord(addrH, addrL)

The function [RomWord\(\)](#) reads a word from rom and returns all 12 bits.

The upper 4 bits are returned in M, the lower 8 bits in W.

No checks are done whether the specified address is valid.

How to use:

```
//in case the address is a constant
#define ADDR 0x123
RomWord(ADDR>>8,ADDR&0xFF);
//in case the address is in a variable
long address = 0x123;
RomWord(address.high8,address.low8);
//capture the data
long value;
value.low8 = W;
value.high8 = MODE; //only if upper 4 bits are needed
```

#### Parameters:

**addrH** High byte of physical rom address (SX18/20/28: 0x000-0x7FF, SX48/52: 0x000-0xFFFF).

**addrL** Low byte of physical rom address.

Definition at line 484 of file memory.h.

## 2.9.3 Function Documentation

### 2.9.3.1 char \_SystemRamAddress (char W)

The function `_SystemRamAddress()` converts a logical `_SystemRam[]` index into a physical ram address. The physical address can be used directly by loading it into FSR and using INDF.

The mapping from logical index to physical address on the SX18/20/28:

- logical -> physical
- 0x00-0x0F -> 0x00-0x0F
- 0x10-0x1F -> 0x10-0x1F
- 0x20-0x2F -> 0x30-0x3F
- 0x30-0x3F -> 0x50-0x5F
- 0x40-0x4F -> 0x70-0x7F
- 0x50-0x5F -> 0x90-0x9F
- 0x60-0x6F -> 0xB0-0xBF
- 0x70-0x7F -> 0xD0-0xDF
- 0x80-0x8F -> 0xF0-0xFF

The mapping from logical index to physical address on the SX48/52:

- logical -> physical
- 0x00-0xFF -> 0x00-0xFF



How to use:

```
addr = _SystemRamAddress(index);
```

**Parameters:**

*W* Logical index into `_SystemRam[]` array (SX18/20/28: 0-143, SX48/52: 0-255).

**Returns:**

Physical ram address.

Definition at line 309 of file memory.h.

### 2.9.3.2 `char _SystemRamIndex (char W)`

The function `_SystemRamIndex()` converts a physical address into a logical `_SystemRam[]` index.

The logical index value can be used to calculate offsets within arrays.

The mapping from physical address to logical index on the SX18/20/28:

- physical -> logical
- 0x00-0x0F -> 0x00-0x0F
- 0x10-0x1F -> 0x10-0x1F
- 0x30-0x3F -> 0x20-0x2F
- 0x50-0x5F -> 0x30-0x3F
- 0x70-0x7F -> 0x40-0x4F
- 0x90-0x9F -> 0x50-0x5F
- 0xB0-0xBF -> 0x60-0x6F
- 0xD0-0xDF -> 0x70-0x7F
- 0xF0-0xFF -> 0x80-0x8F

The mapping from physical address to logical index on the SX48/52:

- physical -> logical
- 0x00-0xFF -> 0x00-0xFF

How to use:

```
index = _SystemRamIndex(addr);
```

**Parameters:**

*W* Physical ram address.

**Returns:**

Logical index into `_SystemRam[]` array.

Definition at line 348 of file memory.h.

### 2.9.3.3 void DDR\_init (void)

The function [DDR\\_init\(\)](#) initializes the shadow DDR registers to 0xFF. This function should be called after [clearRAM\(\)](#) at the start of the `main()` function and before port and pin setup routines.

How to use:

```
DDR_init();
```

Definition at line 50 of file `memory.h`.

### 2.9.3.4 char LargeArrayAddress (char W)

The function [LargeArrayAddress\(\)](#) converts a logical array index into a physical ram address.

The array is specified by the physical address of array element 0.

The physical address can be used directly by loading it into FSR and using INDF.

How to use:

```
_MainTemp = index; //Logical index into the array.  
addr = LargeArrayAddress(base);
```

#### Parameters:

*W* Base, physical ram address of array element 0.

#### Returns:

Physical ram address of array element index.

Definition at line 425 of file `memory.h`.

### 2.9.3.5 char LargeArrayIndex (char W)

The function [LargeArrayIndex\(\)](#) converts a physical ram address into a logical array index.

The array is specified by the physical address of array element 0.

The logical index can be used directly with the array variable.

How to use:

```
_MainTemp = base; //Physical ram address of array element 0  
index = LargeArrayIndex(addr);
```

#### Parameters:

*W* Addr, physical ram address of array element index.

#### Returns:

Logical index into the array.

Definition at line 448 of file `memory.h`.

### 2.9.3.6 void RomCopy (char *dest*, long *source*, char *len*)

The function `RomCopy()` copies bytes from rom to ram. These bytes are the lower 8 bits of the rom words. No checks are done on validity of rom and ram addresses.

Call this function only from mainline code. For a similar function for TASK subroutines see `TaskRomCopy()` in the `Task.h` library.

How to use:

```
RomCopy (dest, source, len);
```

#### Parameters:

*dest* Physical ram address.

*source* Physical rom address (SX18/20/28: 0x000-0x7FF, SX48/52: 0x000-0xFFF).

*len* Number of bytes to copy.

Definition at line 510 of file memory.h.

References `LargeArrayAddress()`, and `RomWord`.

## 2.9.4 Variable Documentation

### 2.9.4.1 shadowDef char \_SystemRam [256] x00

Definition at line 16 of file memory.h.

### 2.9.4.2 char \_IsrBank x0C

Definition at line 19 of file memory.h.

### 2.9.4.3 char \_IsrTemp x0D

Definition at line 18 of file memory.h.

### 2.9.4.4 char \_MainTemp x0E

Definition at line 17 of file memory.h.

### 2.9.4.5 char DDRACOPY xF5

Definition at line 27 of file memory.h.

### 2.9.4.6 char DDRBCOPY xF6

Definition at line 28 of file memory.h.

**2.9.4.7 char DDRCCOPY xF7**

Definition at line 30 of file memory.h.

**2.9.4.8 char DDRDCOPY xF8**

Definition at line 33 of file memory.h.

**2.9.4.9 char DDRECOPY xF9**

Definition at line 34 of file memory.h.

## 2.10 lib/system/portpin.h File Reference

Library for port and pin access.

### Defines

- #define `readPin`(port, pinmask)  
*Read pin.*
- #define `readPinDirection`(port, pinmask)  
*Read pin direction.*
- #define `setPinHigh`(port, pinmask)  
*Make specified port pins high.*
- #define `setPinInput`(port, pinmask)  
*Make specified port pins input.*
- #define `setPinLow`(port, pinmask)  
*Make specified port pins low.*
- #define `setPinOutput`(port, pinmask)  
*Make specified port pins output.*
- #define `setPortLevel`(port, level)  
*Set port level.*
- #define `setPortPullup`(port, pullup)  
*Set port pullup.*
- #define `setPortSchmittTrigger`(port, trigger)  
*Set port schmitt trigger.*
- #define `togglePin`(port, pinmask)  
*Toggle pin.*
- #define `togglePinDirection`(port, pinmask)  
*Toggle pin direction.*
- #define `writePinDirection`(port, pinmask)  
*Write pin direction.*
- #define `writePinLatch`(port, pinmask)  
*Make specified port pins low or high.*
- #define `writePortDirection`(port, direction)  
*Write port direction register.*
- #define `writePortLatch`(port, value)  
*Write port latch register.*

## Functions

- void `DDR_update` (char W)  
*Update chip DDR register.*
- char `readPort` (char W)  
*Read port.*
- char `readPortDirection` (char W)  
*Read port direction register.*

### 2.10.1 Detailed Description

This library contains functions to manipulate ports and pins via variables.

For fixed pins and ports, use ports and pins directly, eg. `RA = 0x12`, `RA.3 = 1`.

The functions are mostly macros. All functions can be used in any call tree (interrupt, mainlevel and tasklevel). Some functions require that a parameter is stored in a global ram location before calling.

Definition in file [portpin.h](#).

### 2.10.2 Define Documentation

#### 2.10.2.1 #define readPin(port, pinmask)

The function `readPin()` reads the specified port input pins (it does not read specified port latch register bits).

A '1' bit in the pinmask specifies the corresponding pins must be extracted. The pinmask MUST be stored in

a global ram location before calling this function. If a single pin is specified, the `Zero_` flag identifies the inverted pin input level. If multiple pins are specified, `W` identifies which pins have a high input.

How to use:

```
_MainTemp = pinmask;
readPin(port, _MainTemp);
pinlevel = !Zero_; //0=low 1=high
```

#### Parameters:

**port** Identifier for port which pins must be set.

You can use `PORT_RA` to `PORT_RE` or the highbyte of a portpin value or portpin constant (`PORTPIN_RA3>>8`).

**pinmask** Specifies which pin to read.

A single pin can be specified, eg. `PORTPIN_RA0`, or an ORed combination

eg. `PORTPIN_RB0 | PORTPIN_RB2 | PORTPIN_RB7` or `PORTPIN_RB` for all pins.

#### Returns:

level at pin input 0 for low, 1 for high.

Definition at line 539 of file portpin.h.

### 2.10.2.2 #define readPinDirection(port, pinmask)

The function `readPinDirection()` reads specified bits of the direction register of the specified port (actually its shadow register). A '1' bit in the pinmask specifies the corresponding pins must be extracted. The pinmask MUST be stored in a global ram location before calling this function. If a single pin is specified, the `Zero_` flag identifies the inverted pin direction. If multiple pins are specified, `W` identifies which pins are inputs.

How to use:

```
_MainTemp = pinmask;
readPinDirection(port, _MainTemp);
pindirection = !Zero_; //0=output 1=input
*
```

#### Parameters:

- port** Identifier for port which pin direction must be read.  
You can use `PORT_RA` to `PORT_RE` or the highbyte of a portpin value or portpin constant (`PORTPIN_RA3>>8`).
- pinmask** Specifies which pin direction to read.  
A single pin can be specified, eg. `PORTPIN_RA0`, or an ORed combination  
eg. `PORTPIN_RB0 | PORTPIN_RB2 | PORTPIN_RB7` or `PORTPIN_RB` for all pins.

#### Returns:

direction Specifies direction for pin 0 = output, 1 = input.

Definition at line 570 of file portpin.h.

### 2.10.2.3 #define setPinHigh(port, pinmask)

The function `setPinHigh()` sets specified bits in the port latch register. The port direction registers are not changed. A '1' bit in the pinmask specifies the corresponding latch register bits must be set. The pinmask MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = pinmask;
setPinHigh(port, _MainTemp);
```

#### Parameters:

- port** Identifier for port which pins must be set.  
You can use `PORT_RA` to `PORT_RE` or the highbyte of a portpin value or portpin constant (`PORTPIN_RA3>>8`).
- pinmask** Specifies which latch register bits to set.  
A single pin can be specified, eg. `PORTPIN_RA0`, or an ORed combination  
eg. `PORTPIN_RB0 | PORTPIN_RB2 | PORTPIN_RB7` or `PORTPIN_RB` for all pins.

Definition at line 471 of file portpin.h.

#### 2.10.2.4 #define setPinInput(port, pinmask)

The function `setPinInput()` sets specified bits in a shadow DDR register and then updates the appropriate chip DDR register. The port latch registers are not changed. A '1' bit in the pinmask specifies the corresponding pins must be set to input. The pinmask MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = pinmask;  
setPinInput(port, _MainTemp);
```

##### Parameters:

**port** Identifier for port which pins must be made inputs.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

**pinmask** Specifies which pin or pins to make inputs.

A single pin can be specified, eg. PORTPIN\_RA0, or an ORed combination eg. PORTPIN\_RB0 | PORTPIN\_RB2 | PORTPIN\_RB7 or PORTPIN\_RB for all pins.

Definition at line 406 of file portpin.h.

#### 2.10.2.5 #define setPinLow(port, pinmask)

The function `setPinLow()` clears specified bits in the port latch register.

The port direction registers are not changed. A '1' bit in the pinmask specifies the corresponding latch register bits must be cleared. The pinmask MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = pinmask;  
setPinLow(port, _MainTemp);
```

##### Parameters:

**port** Identifier for port which pins must be cleared.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

**pinmask** Specifies which latch register bits to clear.

A single pin can be specified, eg. PORTPIN\_RA0, or an ORed combination eg. PORTPIN\_RB0 | PORTPIN\_RB2 | PORTPIN\_RB7 or PORTPIN\_RB for all pins.

Definition at line 439 of file portpin.h.



### 2.10.2.6 #define setPinOutput(port, pinmask)

The function `setPinOutput()` clears specified bits in a shadow DDR register and then updates the appropriate chip DDR register. The port latch registers are not changed. A '1' bit in the pinmask specifies the corresponding pins must be set to output. The pinmask MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = pinmask;
setPinOutput (port, _MainTemp);
```

#### Parameters:

**port** Identifier for port which pins must be made outputs.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

**pinmask** Specifies which pins to make outputs.

A single pin can be specified, eg. PORTPIN\_RA0, or an ORed combination eg. PORTPIN\_RB0 | PORTPIN\_RB2 | PORTPIN\_RB7 or PORTPIN\_RB for all pins.

Definition at line 373 of file portpin.h.

### 2.10.2.7 #define setPortLevel(port, level)

The function `setPortLevel()` writes a value to the port level register.

A '1' bit in the value specifies TTL pin level, a '0' bit specifies CMOS pin level.

The value to write MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = level;
setPortLevel (port, _MainTemp);
```

#### Parameters:

**port** Identifier for port which level register to write.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

**level** Bits specify cmos (2.5V) or ttl (1.4V) level for corresponding pins.

0 = cmos, 1 = ttl.

Definition at line 251 of file portpin.h.

### 2.10.2.8 #define setPortPullup(port, pullup)

The function `setPortPullup()` writes a value to the port pullup register.

A '1' bit in the value specifies no pullup, a '0' bit specifies pullup enabled.

The value to write MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = pullup;
setPortPullup(port, _MainTemp);
```

**Parameters:**

*port* Identifier for port which level register to write.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

*pullup* Bits specify internal pullup resistor settings for corresponding pins.

0 = enabled, 1 = disabled.

Definition at line 291 of file portpin.h.

**2.10.2.9 #define setPortSchmittTrigger(port, trigger)**

The function [setPortSchmittTrigger\(\)](#) writes a value to the port schmitt-trigger register.

A '1' bit in the value specifies no schmitt-trigger input, a '0' bit specifies schmitt-trigger input enabled.

The value to write MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = trigger;
setPortSchmittTrigger(port, _MainTemp);
```

**Parameters:**

*port* Identifier for port which schmitt-trigger register to write.

You can use PORT\_RB to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RB3>>8).

*trigger* Bits specify schmitt-trigger input settings for corresponding pins.

0 = enabled, 1 = disabled.

Definition at line 331 of file portpin.h.

**2.10.2.10 #define togglePin(port, pinmask)**

The function [togglePin\(\)](#) writes the inverted input pin levels to the port latch register.

The port direction registers are not changed. A '1' bit in the pinmask specifies the corresponding latch register bits that must be written. The pinmask MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = pinmask;
togglePin(port, _MainTemp);
```

**Parameters:**

*port* Identifier for port which latch register bits must be written with the inverted input level.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

*pinmask* Specifies which latch register bits to write.

A single pin can be specified, eg. PORTPIN\_RA0, or an ORed combination  
eg. PORTPIN\_RB0 | PORTPIN\_RB2 | PORTPIN\_RB7 or PORTPIN\_RB for all pins.

Definition at line 660 of file portpin.h.

### 2.10.2.11 #define togglePinDirection(port, pinmask)

The function `togglePinDirection()` inverts the specified bits of the port direction register.

The port latch registers are not changed. A '1' bit in the pinmask specifies the corresponding direction register bits that must be inverted. The pinmask MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = pinmask;
togglePinDirection(port, _MainTemp);
```

#### Parameters:

*port* Identifier for port which pin direction bits must be inverted.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

*pinmask* Specifies which direction register bits to invert.

A single pin can be specified, eg. PORTPIN\_RA0, or an ORed combination  
eg. PORTPIN\_RB0 | PORTPIN\_RB2 | PORTPIN\_RB7 or PORTPIN\_RB for all pins.

Definition at line 631 of file portpin.h.

### 2.10.2.12 #define writePinDirection(port, pinmask)

This function `writePinDirection()` sets or clears specified bits in a shadow DDR register and then updates the appropriate chip DDR register. The port latch registers are not changed. A '1' bit in the pinmask specifies

the corresponding direction register bits must be set or cleared. The pinmask MUST be stored in a global ram

location before calling this function. The carry value specifies whether direction register bits are set or cleared.

How to use:

```
Carry = direction; //0=output, 1=input
_MainTemp = pinmask;
writePinDirection(port, _MainTemp);
```

#### Parameters:

*port* Identifier for port which pin direction must be read.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

*pinmask* Specifies which pin direction to write.

A single pin can be specified, eg. PORTPIN\_RA0, or an ORed combination  
eg. PORTPIN\_RB0 | PORTPIN\_RB2 | PORTPIN\_RB7 or PORTPIN\_RB for all pins.

Definition at line 599 of file portpin.h.

### 2.10.2.13 #define writePinLatch(port, pinmask)

The function [writePinLatch\(\)](#) sets or clears specified bits in the port latch register.

The port direction registers are not changed. A '1' bit in the pinmask specifies the corresponding latch register bits must be set or cleared. The pinmask MUST be stored in a global ram location before calling this function. The carry value specifies wether latch register bits are set or cleared.

How to use:

```
Carry = level; //0=low, 1=high
_MainTemp = pinmask;
writePin(port, _MainTemp);
```

#### Parameters:

*port* Identifier for port which pins must be set or cleared.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

*pinmask* Specifies which latch register bits to set or clear (determined by carry).

A single pin can be specified, eg. PORTPIN\_RA0, or an ORed combination  
eg. PORTPIN\_RB0 | PORTPIN\_RB2 | PORTPIN\_RB7 or PORTPIN\_RB for all pins

Definition at line 505 of file portpin.h.

### 2.10.2.14 #define writePortDirection(port, direction)

The function [writePortDirection\(\)](#) writes a value to the port direction register.

The port latch register is not changed. A '1' bit in the value specifies an input pin, a '0' bit specifies an output pin. The value to write MUST be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = direction;
writePortDirection(port, _MainTemp);
```

#### Parameters:

*port* Identifier for port which direction register to write.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

*direction* Bits specify direction for corresponding pins.

0 = output, 1 = input.

Definition at line 132 of file portpin.h.

### 2.10.2.15 #define writePortLatch(port, value)

The function `writePortLatch()` writes a value to the port latch register.

The port direction register is not changed. A '1' bit in the value specifies a high pin output level, a '0' bit specifies a low pin output level. The value to write **MUST** be stored in a global ram location before calling this function.

How to use:

```
_MainTemp = value;
writePortLatch(port, _MainTemp);
```

#### Parameters:

**port** Identifier for port which latch to write.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

**value** Bits specify output level for corresponding pins.

0 = low, 1 = high.

Definition at line 193 of file portpin.h.

## 2.10.3 Function Documentation

### 2.10.3.1 void DDR\_update (char W)

The function `DDR_update()` writes the DDR shadow register for the specified port to the appropriate chip DDR register. It is used by the pin and port direction functions. This function prevents possible glitches on port pins (in case the interrupt routine also accesses the DDR registers), by inserting a few NOPs when a RTCC rollover interrupt would occur between reading the DDR shadow register and updating the chip DDR register.

#### Parameters:

**W** Port identifier (PORT\_RA to PORT\_RE).

It is also possible to use the highbyte of a portpin value, eg. PORTPIN\_RA3>>8.

Definition at line 35 of file portpin.h.

References PORT\_RA, PORT\_RB, PORT\_RC, PORT\_RD, and PORT\_RE.

### 2.10.3.2 char readPort (char W)

The function `readPort()` reads the port pins (it does not read the port latch register).

A '1' bit in the value specifies a high level, a '0' bit specifies a low level.

How to use:

```
value = readPort(port);
```

**Parameters:**

*W* Identifier for port which pins to read.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

**Returns:**

value Bits specify levels at corresponding pins.

0 = low, 1 = high.

Definition at line 221 of file portpin.h.

**2.10.3.3 char readPortDirection (char *W*)**

The function [readPortDirection\(\)](#) reads the direction register of the specified port (actually its shadow register). A '1' bit in the value specifies an input pin, a '0' bit specifies an output pin.

How to use:

```
direction = readPortDirection(port);
```

**Parameters:**

*W* Identifier for port which direction register to read.

You can use PORT\_RA to PORT\_RE or the highbyte of a portpin value or portpin constant (PORTPIN\_RA3>>8).

**Returns:**

direction Bits specify direction for corresponding pins.

0 = output, 1 = input.

Definition at line 162 of file portpin.h.

## 2.11 lib/taskswitching/Task.h File Reference

Library for taskswitching support.

### Defines

- #define `TASK` void
- #define `Task_ISR`  
*The interrupt part of the task scheduler.*
- #define `TASKINTERVAL` (TASKS)
- #define `TASKKERNEL` 0xF0
- #define `TASKRUNONCE` 0x40
- #define `TaskSet`(slot, taskid, interval)  
*Install task in tasklist.*
- #define `TASKSTART` 0x80

### Functions

- void `TaskDisable` (void)  
*Disable the task scheduler.*
- void `TaskEnable` (void)  
*Enable the task scheduler.*
- void `TaskInit` (void)  
*Initialize the task scheduler.*
- void `TaskReschedule` (char W)  
*Reschedule a task.*
- void `TaskRomCopy` (char dest, long source, char len)  
*Copy bytes from rom to ram.*
- void `TaskSchedule` (void)  
*The mainlevel part of the task scheduler.*
- char `TaskSlot` (char W)  
*Locate slot for task subroutine.*
- char `TaskSlotAvailable` (void)  
*Locate available task slot.*
- void `TaskSlotStart` (char W)  
*Start a task.*
- void `TaskSlotStop` (char W)  
*Stop a task.*

- void `TaskStart` (char W)  
*Start a task.*
- void `TaskStop` (char W)  
*Stop a task.*
- void `TaskSwitch` (char W)  
*Taskswitch handler (extern).*
- char `TaskTimer` (void)  
*Get task timer value.*

### 2.11.1 Detailed Description

The taskswitch mechanism is NOT multitasking. It allows subroutines to run automatically at specified intervals. It can best be compared to the Quick Basic ON TIMER GOSUB command. The interrupt routine monitors when it is time to run a task. Then a contextswitch is done, so that after the interrupt routine exits, a task routine is executed, rather than the mainline code that was interrupted. The interrupts remain active while the task runs. Only after the task exits, will the mainline code continue (after another contextswitch). It is therefore imperative that a task never blocks or waits for an event. If it must wait, let it set a flag and return. The next time it runs, it can check the flag. The SX secret instructions are used to perform the contextswitch.

For clarity, TASK is defined as void. It allows you to use

```
TASK myTask()
{
}
```

Note that task routines appear to CC1B as part of the interrupt call tree, although the task routines themselves execute as mainlevel code. You should never call a TASK subroutine from mainline code.

Here are some guide 'rules' to determine the correct tasks parameters.

The goal is to LET EACH TASK START ON TIME.

For each task to run on time, each task must return within 1 tasktick and the minimum interval must be equal to the number of tasks. To guarantee a minimum percentage of CPU cycles for the mainline code, you must use an interval greater than the number of tasks.

```
#define TASKS 4
#define TASKINTERVAL 5
```

This guarantees  $((5-4)/5) * 100\% = 20\%$  of CPU cycles is used for mainline code.

For each task to run on time, the individual task intervals must be a multiple of TASKINTERVAL.

If you specify for some task an interval of 2\*TASKINTERVAL, then each 2nd timeslice for



that task is not used to run the task but used for mainline code. If a task does not return within 1 tasktick, either split up the task in smaller pieces (state machine) or increase the TASKTICK value. So the rule is:

**All tasks must have an interval equal to  $K \cdot \text{TASKINTERVAL}$  and must return within 1 tasktick and should start 1 tasktick apart. This ensures tasks do not overlap and start on time.**

You are not bound by the above rule. You can set the interval to any non-zero value (1-255) but then it may happen that tasks are at some point scheduled for the same time and then the tasks are not deterministic. Tasks scheduled for the same time are simply executed after each other without returning to the mainline code inbetween.

If  $K > 1$  then a task does not use all its timeslices. Unused timeslices are not available to other tasks but are used for mainline code. If a task takes too much time, a statemachine can be used to make a task return in time. If a task must run at a lower rate than set by its interval parameter, the task can decrement a counter and return immediately if it has not reached 0. Since the interval parameter is an 8bit value, the highest K calculates from  $K \cdot \text{TASKINTERVAL} = 255$ .

This yields  $K_{\max} = \text{int}(255/\text{TASKINTERVAL})$

INTPERIOD determines the interrupt tick:  $\text{isrtick} = \text{INTPERIOD}/\text{CPUFREQ}$

TASKTICK determines the task tick:  $\text{tasktick} = \text{TASKTICK} \cdot \text{isrtick}$

The maximum value for TASKTICK is 65279.

TASKINTERVAL must be  $\geq$  TASKS. If you don't define TASKINTERVAL then TASKINTERVAL will be set to TASKS.

Example: suppose you want to run a task 1000 times per second, with given CPUFREQ=20\_000\_000 and INTPERIOD=217 and 4 tasks running:

$\text{isrtick} = \text{INTPERIOD}/\text{CPUFREQ} = 217/20\_000\_000 = 10.85 \text{ uSec}$

$\text{tasktick} = \text{TASKTICK} \cdot 10.85 \text{ uSec} = (1/1000) \text{ Sec} / 4 = 250 \text{ uSec}$

$\text{TASKTICK} = 250/10.85 = 23$  for interval = 4.

Normally the task that must run at the highest rate determines TASKTICK and task intervals are adjusted for tasks that run at a lower rate.

Definition in file [Task.h](#).

## 2.11.2 Define Documentation

### 2.11.2.1 #define TASK void

Definition at line 112 of file Task.h.

### 2.11.2.2 #define Task\_ISR

The function [Task\\_ISR\(\)](#) must run at the end of the interrupt routine. It must not be called anywhere else. Once installed, if a task must run, it redirects program control to [TaskSchedule\(\)](#).

How to use:

```

interrupt iServer()
{
  #if defined _CHIP_SX18_ || defined _CHIP_SX20_ || defined _CHIP_SX28_
  _IsrMode = MODE; //save M
  #endif
  //-----
  //other interrupt code here
  //-----
  Task_ISR; //interrupt part of the task scheduler
  #if defined _CHIP_SX18_ || defined _CHIP_SX20_ || defined _CHIP_SX28_
  MODE = _IsrMode; //restore M
  #endif
  W = -INTPERIOD;
  retiw();
}

```

Definition at line 674 of file Task.h.

### 2.11.2.3 #define TASKINTERVAL (TASKS)

Definition at line 97 of file Task.h.

### 2.11.2.4 #define TASKKERNEL 0xF0

Definition at line 122 of file Task.h.

### 2.11.2.5 #define TASKRUNONCE 0x40

Definition at line 114 of file Task.h.

### 2.11.2.6 #define TaskSet(slot, taskid, interval)

The function [TaskSet\(\)](#) sets up the task parameters in the TaskList.

The parameter taskid may be ORed with TASKSTART (to autostart task) and TASKRUNONCE (so the task only runs once).

#### Parameters:

*slot* Index in TaskList (0 to TASKS-1).

*taskid* Identifier for task subroutine (may be ORed with TASKSTART and/or TASKRUNONCE).

*interval* Specifies in taskticks ( $\geq 1$ ) how frequently the task runs. Do not use value 0.

Definition at line 242 of file Task.h.

### 2.11.2.7 #define TASKSTART 0x80

Definition at line 113 of file Task.h.

## 2.11.3 Function Documentation

### 2.11.3.1 void TaskDisable (void)

The function [TaskDisable\(\)](#) prevents all tasks from running when called from mainline code. It pauses the task timer. If however it is called from a task subroutine then this subroutine does not need to return within 1 tasktick. This may be handy in the case of a high priority event. When used in a task subroutine, the task should call [TaskEnable\(\)](#) before returning to allow other tasks to run, otherwise no other task will run after returning to mainline code.

How to use:

```
TASK myTask()
{
    if (highPriority) {
        TaskDisable();
        HandleEvent(); //this may take longer than 1 tasktick
        TaskEnable();
    }
}
```

Definition at line 206 of file Task.h.

### 2.11.3.2 void TaskEnable (void)

The function [TaskEnable\(\)](#) allows tasks to run. It starts the task timer. It does not reset the task timer, it simply continues to count, so to tasks it appears time was frozen while [TaskDisable\(\)](#) was active. After calling this function, all tasks that are scheduled to run, will be run.

```
TaskEnable();
```

Definition at line 224 of file Task.h.

### 2.11.3.3 void TaskInit (void)

The function [TaskInit\(\)](#) copies the task initialization data from rom to ram. It is to be called once from mainlevel code only and it must be called before any of the other task functions.

How to use:

```
TaskInit();
```

Definition at line 176 of file Task.h.

References [RomCopy\(\)](#).

### 2.11.3.4 void TaskReschedule (char W)

The function [TaskReschedule\(\)](#) reschedules the task this function is called from.

When called from mainline code nothing happens. It is used to allow a task to change the rate it is run, due to some condition, for example the arrival of data. The task will be rescheduled using the parameter rather than its interval value. The interval value however is not changed.

How to use:

```
TASK mytask()
{
  //some code
  TaskReschedule(12); //run this task again, 12 taskticks after this task returns
}
```

**Parameters:**

- W** Number of taskticks after which the task runs again once it returns.  
A value of 0 will use the task interval value for rescheduling.

Definition at line 458 of file Task.h.

**2.11.3.5 void TaskRomCopy (char *dest*, long *source*, char *len*)**

The function [TaskRomCopy\(\)](#) copies bytes from rom to ram. These bytes are the lower 8 bits of the rom words. No checks are done on validity of rom and ram addresses.

Call this function only from TASK subroutines. For a similar function for mainline code see [RomCopy\(\)](#) in the [memory.h](#) library.

How to use:

```
TaskRomCopy (dest, source, len);
```

**Parameters:**

- dest*** Physical ram address.
- source*** Physical rom address (SX18/20/28: 0x000-0x7FF, SX48/52: 0x000-0xFFF).
- len*** Number of bytes to copy.

Definition at line 482 of file Task.h.

References [LargeArrayAddress\(\)](#), and [RomWord](#).

**2.11.3.6 void TaskSchedule (void)**

The function [TaskSchedule\(\)](#) is called by the interrupt part of the task scheduler.

It should never be called from anywhere else. It executes as mainlevel code.

This function calls the function [TaskSwitch](#) that must be supplied by the application.

It is important to realize that this function in effect interrupts the mainline code.

The application must supply a function [TaskSwitch\(\)](#) that must look like this:

```
#define MYTASK1 12 //unique identifiers for tasks (1 to 63)
#define MYTASK2 25

TASK myTask1 ()
{
}

TASK myTask2 ()
{
}

void TaskSwitch(char W)
{
    switch (W & 0x3F) { //call task, returns within 1 tasktick
        case MYTASK1: myTask1(); break;
        case MYTASK2: myTask2(); break;
        //more tasks here
    }
}
```

Note that the functions [TaskSwitch\(\)](#) and the task subroutines all execute as mainlevel code. They do not consume interrupt cycles.

Definition at line 557 of file Task.h.

References TASKKERNEL, and TaskSwitch().

### 2.11.3.7 char TaskSlot (char W)

The function [TaskSlot\(\)](#) locates in which slot a specific task is installed.

If the task is found, its slot id is returned, otherwise 0xFF.

#### Parameters:

*W* Task identifier.

#### Returns:

Slot in which task is installed, or 0xFF if task not found (eg. task not set).

Definition at line 259 of file Task.h.

### 2.11.3.8 char TaskSlotAvailable (void)

The function [TaskSlotAvailable\(\)](#) locates a free task slot.

If available, its id is returned, otherwise 0xFF.

#### Returns:

Slot id for available slot, or 0xFF (no slot available).

Definition at line 295 of file Task.h.

### 2.11.3.9 void TaskSlotStart (char W)

The function [TaskSlotStart\(\)](#) enables the task that is installed in the specified slot.

If no task is specified for that slot nothing happens, otherwise the task will run at the time it is scheduled for.

**Parameters:**

*W* Slot id (0 to TASKS-1).

Definition at line 409 of file Task.h.

**2.11.3.10 void TaskSlotStop (char *W*)**

The function [TaskSlotStop\(\)](#) disables the task that is installed in the specified slot. The task will not run until it is enabled again by [TaskStart\(\)](#) or [TaskSlotStart\(\)](#), but it will continue to reschedule according to its interval parameter, so not to upset the task sequence.

**Parameters:**

*W* Slot id (0 to TASKS-1).

Definition at line 428 of file Task.h.

**2.11.3.11 void TaskStart (char *W*)**

The function [TaskStart\(\)](#) enables a task that has been installed in the tasklist. If the task is not in the tasklist nothing happens, otherwise the task will run at the time it is scheduled for.

**Parameters:**

*W* Task identifier.

Definition at line 366 of file Task.h.

References [TaskSlot\(\)](#).

**2.11.3.12 void TaskStop (char *W*)**

The function [TaskStop\(\)](#) disables a task that has been installed in the tasklist. If the task is not in the tasklist nothing happens, otherwise the task will not run until it is enabled again by [TaskStart\(\)](#) or [TaskSlotStart\(\)](#), but it will continue to reschedule according to its interval parameter, so not to upset the task sequence.

**Parameters:**

*W* Task identifier.

Definition at line 388 of file Task.h.

References [TaskSlot\(\)](#).

### 2.11.3.13 void TaskSwitch (char W)

The function `TaskSwitch()` must be supplied by the application.

A typical layout for this function:

```
#define TXDEQUEUE 1 //unique identifiers for task subroutines
#define RXENQUEUE 2
void TaskSwitch(char W)
{
    switch (W & 0x3F) { //call task, returns within 1 tasktick
        case TXDEQUEUE: TxDequeue(); break; //TxDequeue has identifier 0
        case RXENQUEUE: RxEnqueue(); break; //RxEnqueue has identifier 1
    }
}
```

#### Parameters:

*W* Task identifier.

### 2.11.3.14 char TaskTimer (void)

The function `TaskTimer()` returns the current task timer count. This is an 8bit value.

The counter simply increments every tasktick, except when `TaskDisable()` is called.

Then the timer is paused until `TaskEnable()` is called.

#### Returns:

Current task timer value.

Definition at line 330 of file `Task.h`.

## 2.12 lib/text/character/ctype.h File Reference

Library for character functions.

### Functions

- bit [isalnum](#) (char ch)  
*Test if character is alphanumeric.*
- bit [isalpha](#) (char ch)  
*Test if character is alphabetic.*
- bit [isascii](#) (char ch)  
*Test if character is an ascii character.*
- bit [isctrl](#) (char ch)  
*Test if character is a control character.*
- bit [isdigit](#) (char ch)  
*Test if character is a digit.*
- bit [isgraph](#) (char ch)  
*Test if character is a printable character other than a space.*
- bit [islower](#) (char ch)  
*Test if character is a lowercase alphabetic.*
- bit [isprint](#) (char ch)  
*Test if character is a printable character (32-126).*
- bit [ispunct](#) (char ch)  
*Test if character is a punctuation character.*
- bit [isspace](#) (char ch)  
*Test if character is a space, tab or newline.*
- bit [isupper](#) (char ch)  
*Test if character is an uppercase alphabetic.*
- bit [isxdigit](#) (char ch)  
*Test if character is a hexadecimal digit.*
- char [toascii](#) (char ch)  
*Convert character to equivalent ascii value.*
- char [tolower](#) (char ch)  
*Convert character to lowercase.*
- char [toupper](#) (char ch)  
*Convert character to uppercase.*



## 2.12.1 Detailed Description

This library provides character functions.

Definition in file [ctype.h](#).

## 2.12.2 Function Documentation

### 2.12.2.1 bit isalnum (char *ch*)

Test if character is alphanumeric (a-z, A-Z or 0-9).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is alphanumeric.

Definition at line 80 of file [ctype.h](#).

References [isalpha\(\)](#), and [isdigit\(\)](#).

### 2.12.2.2 bit isalpha (char *ch*)

Test if character is alphabetic (a-z or A-Z).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is alphabetic.

Definition at line 24 of file [ctype.h](#).

### 2.12.2.3 bit isascii (char *ch*)

Test if character is an ascii character (0-127).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is an ASCII character (0-127).

Definition at line 38 of file [ctype.h](#).

**2.12.2.4 bit iscntrl (char *ch*)**

Test if character is a control character (0-31 or 127).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is a control character.

Definition at line 52 of file ctype.h.

**2.12.2.5 bit isdigit (char *ch*)**

Test if character is a digit (0-9).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is a digit.

Definition at line 66 of file ctype.h.

**2.12.2.6 bit isgraph (char *ch*)**

Test if character is a printable character other than a space (33-126).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is a printable character other than a space.

Definition at line 96 of file ctype.h.

**2.12.2.7 bit islower (char *ch*)**

Test if character is a lowercase alphabetic (a-z).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is a lowercase alphabetic.

Definition at line 110 of file ctype.h.

### 2.12.2.8 bit isprint (char *ch*)

Test if character is a printable character (32-126).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is a printable character.

Definition at line 124 of file ctype.h.

### 2.12.2.9 bit ispunct (char *ch*)

Test if character is a punctuation character (all but control and alphanumeric).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is a punctuation character.

Definition at line 138 of file ctype.h.

References `isalnum()`, and `isctrl()`.

### 2.12.2.10 bit isspace (char *ch*)

Test if character is a space, tab or newline.

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is a space, tab or newline.

Definition at line 152 of file ctype.h.

### 2.12.2.11 bit isupper (char *ch*)

Test if character is an uppercase alphabetic (A-Z).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is an uppercase alphabetic.

Definition at line 166 of file ctype.h.

**2.12.2.12 bit isxdigit (char *ch*)**

Test if character is a hexadecimal digit (0-9, A-F, a-f).

**Parameters:**

*ch* Character to be tested.

**Returns:**

True if *ch* is a hexadecimal digit.

Definition at line 180 of file ctype.h.

**2.12.2.13 char toascii (char *ch*)**

Convert character to equivalent ascii value.

**Parameters:**

*ch* Character to be converted.

**Returns:**

ascii equivalent of *ch*.

Definition at line 195 of file ctype.h.

**2.12.2.14 char tolower (char *ch*)**

Convert character to lowercase.

**Parameters:**

*ch* Character to be converted.

**Returns:**

lowercase of *ch* if uppercase, else *ch*.

Definition at line 209 of file ctype.h.

**2.12.2.15 char toupper (char *ch*)**

Convert character to uppercase.

**Parameters:**

*ch* Character to be converted.

**Returns:**

uppercase of *ch* if it is lowercase, else *ch*.

Definition at line 224 of file ctype.h.

## 2.13 lib/text/conversion/stdlib.h File Reference

Library for string conversion support.

```
#include <system/memory.h>
#include <text/character/ctype.h>
#include <text/string/string.h>
```

### Functions

- int [abs](#) (int nbr)  
*Get absolute value of signed integer.*
- int [atoi](#) (char \*s)  
*Convert signed decimal string to integer.*
- int [atoi](#)b (char \*s, int b)  
*Convert unsigned decimal string to integer for base.*
- int [dtoi](#) (char \*decstr, int \*nbr)  
*Convert signed decimal string to integer.*
- char \* [itoa](#) (int n, char \*s)  
*Convert integer to signed decimal string.*
- char \* [itoab](#) (int n, char \*s, int b)  
*Convert integer to unsigned decimal string for base.*
- char \* [itod](#) (int nbr, char \*str, int sz)  
*Convert integer to signed decimal string.*
- char \* [itoo](#) (int nbr, char \*str, int sz)  
*Convert integer to octal string.*
- char \* [itou](#) (int nbr, char \*str, int sz)  
*Convert integer to unsigned decimal string.*
- char \* [itox](#) (int nbr, char \*str, int sz)  
*Convert integer to hexadecimal string.*
- char \* [left](#) (char \*str)  
*Left adjust and null-terminate a string.*
- int [otoi](#) (char \*octstr, int \*nbr)  
*Convert unsigned octal string to integer.*
- char \* [pad](#) (char \*dest, char ch, int n)  
*Place n occurrences of ch at dest.*

- char \* [reverse](#) (char \*s)  
*Reverse string in place.*
- int [sign](#) (int nbr)  
*Get sign of integer.*
- int [utoi](#) (char \*decstr, int \*nbr)  
*Convert unsigned decimal string to integer.*
- int [xtoi](#) (char \*hexstr, int \*nbr)  
*Convert hexadecimal string to integer.*

### 2.13.1 Detailed Description

This library provides functions for string conversions.

Definition in file [stdlib.h](#).

### 2.13.2 Function Documentation

#### 2.13.2.1 int abs (int *nbr*)

Get absolute value of signed integer.

**Parameters:**

*nbr* Number to convert.

**Returns:**

Absolute value of *nbr*.

Definition at line 27 of file [stdlib.h](#).

#### 2.13.2.2 int atoi (char \* *s*)

Convert signed decimal string to integer.

**Parameters:**

*s* Address of signed decimal string in ram.

**Returns:**

Signed integer value.

Definition at line 157 of file [stdlib.h](#).

References [isdigit\(\)](#), and [isspace\(\)](#).

### 2.13.2.3 int atoib (char \* s, int b)

Convert signed decimal string to integer using base b.

This is a non-standard function.

#### Parameters:

*s* Address of unsigned decimal string in ram.

*b* Base for conversion: (2=binary,8=octal,10=decimal,16=hexadecimal).

#### Returns:

Signed integer value.

Definition at line 188 of file stdlib.h.

References isspace().

### 2.13.2.4 int dtoid (char \* decstr, int \* nbr)

Convert signed decimal string to integer.

#### Parameters:

*decstr* Address of signed decimal string in ram.

*nbr* Address of result integer.

#### Returns:

Number of characters read from decstr, -1 if error.

Definition at line 69 of file stdlib.h.

References utoi().

### 2.13.2.5 char\* itoa (int n, char \* s)

Convert integer to signed decimal string.

#### Parameters:

*n* Signed value.

*s* Address of character array in ram.

#### Returns:

str.

Definition at line 341 of file stdlib.h.

References reverse().

**2.13.2.6 char\* itoab (int *n*, char \* *s*, int *b*)**

Convert integer to unsigned decimal string for base *b*.

**Parameters:**

- n* Unsigned value.
- s* Address of character array in ram.
- b* Base (2=binary, 8=octal, 10=decimal, 16=hexadecimal),

**Returns:**

*str*.

Definition at line 371 of file stdlib.h.

References reverse().

**2.13.2.7 char\* itod (int *nbr*, char \* *str*, int *sz*)**

Convert integer to signed decimal string.

**Parameters:**

- nbr* Signed value.
- str* Address of character array in ram.
- sz* Option,
  - sz* > 0: terminate with null byte, *sz* = 0: find end of string, *sz* < 0: use last byte for data.

**Returns:**

*str*.

Definition at line 219 of file stdlib.h.

**2.13.2.8 char\* itoo (int *nbr*, char \* *str*, int *sz*)**

Convert integer to octal string.

**Parameters:**

- nbr* Signed value.
- str* Address of character array in ram.
- sz* Option,
  - sz* > 0: terminate with null byte, *sz* = 0: find end of string, *sz* < 0: use last byte for data.

**Returns:**

*str*.

Definition at line 251 of file stdlib.h.



### 2.13.2.9 char\* itou (int *nbr*, char \* *str*, int *sz*)

Convert integer to unsigned decimal string.

#### Parameters:

*nbr* Unsigned value.

*str* Address of character array in ram.

*sz* Option,

*sz* > 0: terminate with null byte, *sz* = 0: find end of string, *sz* < 0: use last byte for data.

#### Returns:

*str*.

Definition at line 280 of file stdlib.h.

### 2.13.2.10 char\* itox (int *nbr*, char \* *str*, int *sz*)

Convert integer to hexadecimal string.

#### Parameters:

*nbr* Unsigned value.

*str* Address of character array in ram.

*sz* Option,

*sz* > 0: terminate with null byte, *sz* = 0: find end of string, *sz* < 0: use last byte for data.

#### Returns:

*str*.

Definition at line 313 of file stdlib.h.

### 2.13.2.11 char\* left (char \* *str*)

Left adjust and null-terminate a string.

#### Parameters:

*str* String to adjust.

#### Returns:

*str*.

Definition at line 401 of file stdlib.h.

**2.13.2.12 int otoi (char \* octstr, int \* nbr)**

Convert unsigned octal string to integer.

**Parameters:**

*octstr* Address of unsigned octal string in ram.

*nbr* Address of result integer.

**Returns:**

Number of characters read from octstr, -1 if error.

Definition at line 97 of file stdlib.h.

**2.13.2.13 char\* pad (char \* dest, char ch, int n)**

Place n occurrences of ch at dest.

**Parameters:**

*dest* String in which to place ch.

*ch* Character to place in dest.

*n* Number of times to place ch in dest.

**Returns:**

dest.

Definition at line 427 of file stdlib.h.

**2.13.2.14 char\* reverse (char \* s)**

Reverse string in place.

**Parameters:**

*s* String to reverse.

**Returns:**

s.

Definition at line 445 of file stdlib.h.

References strlen().

**2.13.2.15 int sign (int nbr)**

Get sign of integer.

**Parameters:**

*nbr* Signed integer.

**Returns:**

-1, 0, +1 depending on the sign of *nbr*.

Definition at line 470 of file `stdlib.h`.

**2.13.2.16 int utoi (char \* *decstr*, int \* *nbr*)**

Convert unsigned decimal string to integer.

**Parameters:**

*decstr* Address of unsigned decimal string in ram.

*nbr* Address of result integer.

**Returns:**

Number of characters read from *decstr*, -1 if error.

Definition at line 43 of file `stdlib.h`.

**2.13.2.17 int xtoi (char \* *hexstr*, int \* *nbr*)**

Convert hexadecimal string to integer.

**Parameters:**

*hexstr* Address of hexadecimal string in ram.

*nbr* Address of result integer.

**Returns:**

Number of characters read from *hexstr*, -1 if error.

Definition at line 123 of file `stdlib.h`.

## 2.14 lib/text/string/cstring.h File Reference

Library for constant string support.

```
#include <system/memory.h>
#include <text/string/string.h>
```

### Functions

- char \* [cstreat](#) (char \*s, const char \*t)  
*Concatenate constant string to string.*
- const char \* [cstrchr](#) (const char \*s, char c)  
*Locate character in constant string.*
- int [cstrcmp](#) (char \*s, const char \*t)  
*Compare a string against a constant string.*
- char \* [cstrcpy](#) (char \*s, const char \*t)  
*Copy constant string to string.*
- long [cstrlen](#) (const char \*s)  
*Calculate the length of a constant string.*
- char \* [cstrncat](#) (char \*s, const char \*t, int len)  
*Concatenate constant string to string for specified number of characters.*
- int [cstrncmp](#) (char \*s, const char \*t, int len)  
*Compare a string against a constant string for up to a specified number of bytes.*
- char \* [cstrncpy](#) (char \*s, const char \*t, int len)  
*Copy constant string to string for specified number of characters.*
- const char \* [cstrchr](#) (const char \*s, char c)  
*Locate character in constant string.*

### 2.14.1 Detailed Description

This library provides functions for strings located in rom.

Unlike library [dstring.h](#) that uses long for pointers, this

library uses const char \* for pointers.

Definition in file [cstring.h](#).

## 2.14.2 Function Documentation

### 2.14.2.1 `char* cstrcat (char * s, const char * t)`

The `cstrcat()` function appends the `t` string to the `s` string overwriting the `'\0'` character at the end of `s`, and then adds a terminating `'\0'` character. The strings may not overlap, and the `s` string must have enough space for the result.

#### Parameters:

- `s` Address of string in ram.
- `t` Address of string in rom.

#### Returns:

The `cstrcat()` function returns a pointer to the destination string `s`.

Definition at line 52 of file `cstring.h`.

References `strlen()`.

### 2.14.2.2 `const char* cstrchr (const char * s, char c)`

The `cstrchr()` function searches a constant string for the presence of a specific character.

#### Parameters:

- `s` Address of string in rom.
- `c` Character to locate.

#### Returns:

The `cstrchr()` function returns a pointer to the matched character or `NULL` if the character is not found.

Definition at line 77 of file `cstring.h`.

### 2.14.2.3 `int strcmp (char * s, const char * t)`

The `strcmp()` function compares the two strings `s` and `t`.

It returns an integer less than, equal to, or greater than zero if `s` is found, respectively, to be less than, to match, or be greater than `t`.

#### Parameters:

- `s` Address of string in ram.
- `t` Address of string in rom.

#### Returns:

The `strcmp()` function returns an integer less than, equal to, or greater than zero if `s` is found, respectively, to be less than, to match, or be greater than `t`.

Definition at line 100 of file `cstring.h`.

#### 2.14.2.4 `char* strcpy (char * s, const char * t)`

The `strcpy()` function copies the string pointed to by `t` (including the terminating `'\0'` character) to the array pointed to by `s`. The strings may not overlap, and the destination string `s` must be large enough to receive the copy.

##### Parameters:

- `s` Address of string in ram.
- `t` Address of string in rom.

##### Returns:

The `strcpy()` function returns a pointer to the destination string `s`.

Definition at line 129 of file `cstring.h`.

#### 2.14.2.5 `long strlen (const char * s)`

The `strlen()` function calculates the length of the constant string `s`, not including the terminating `'\0'` character.

##### Parameters:

- `s` Address of string in rom.

##### Returns:

The `strlen()` function returns the length of constant string `s`.

Definition at line 29 of file `cstring.h`.

#### 2.14.2.6 `char* strncpy (char * s, const char * t, int len)`

The `strncpy()` function appends up to the specified number of characters from the `t` string to the `s` string overwriting the `'\0'` character at the end of `s`, and then adds a terminating `'\0'` character.

The strings may not overlap, and the `s` string must have enough space for the result.

##### Parameters:

- `s` Address of string in ram.
- `t` Address of string in rom.
- `len` Number of characters to copy.

##### Returns:

The `strncpy()` function returns a pointer to the destination string `s`.

Definition at line 156 of file `cstring.h`.

References `strlen()`.

### 2.14.2.7 `int cstrncmp (char * s, const char * t, int len)`

The `cstrncmp()` function compares the two strings `s` and `t` for the first (at most) `n` characters. It returns an integer less than, equal to, or greater than zero if `s` is (or the first `n` bytes thereof) found, respectively, to be less than, to match, or be greater than `t`.

#### Parameters:

- `s` Address of string in ram.
- `t` Address of string in rom.
- `len` Number of characters to compare.

#### Returns:

The `cstrncmp()` function returns an integer less than, equal to, or greater than zero if `s` (or the first `n` bytes thereof) is found, respectively, to be less than, to match, or be greater than `t`.

Definition at line 189 of file `cstring.h`.

### 2.14.2.8 `char* cstrncpy (char * s, const char * t, int len)`

The `cstrncpy()` function copies up to the specified number of characters from the `t` string (including the terminating ‘`\0`’ character) to the array pointed to by `s`. The strings may not overlap, and the destination string `s` must be large enough to receive the copy.

#### Parameters:

- `s` Address of string in ram.
- `t` Address of string in rom.
- `len` Number of bytes to copy.

#### Returns:

The `cstrncpy()` function returns a pointer to the destination string `s`.

Definition at line 219 of file `cstring.h`.

### 2.14.2.9 `const char* cstrrchr (const char * s, char c)`

The `cstrrchr()` function searches a constant string for the presence of a specific character.

#### Parameters:

- `s` Address of string in rom.
- `c` Character to locate.

#### Returns:

The `cstrrchr()` function returns a pointer to the last occurrence of the character `c` in the string `s`.

Definition at line 254 of file `cstring.h`.

## 2.15 lib/text/string/dstring.h File Reference

Library for constant string support.

```
#include <system/memory.h>
#include <text/string/string.h>
```

### Functions

- char \* [dstreat](#) (char \*s, long t)  
*Concatenate constant string to string.*
- long [dstrchr](#) (long s, char c)  
*Locate character in constant string.*
- int [dstncmp](#) (char \*s, long t)  
*Compare a string against a constant string.*
- char \* [dstncpy](#) (char \*s, long t)  
*Copy constant string to string.*
- long [dstrlen](#) (long s)  
*Calculate the length of a constant string.*
- char \* [dstncat](#) (char \*s, long t, int len)  
*Concatenate constant string to string for specified number of characters.*
- int [dstncmp](#) (char \*s, long t, int len)  
*Compare a string against a constant string for up to a specified number of bytes.*
- char \* [dstncpy](#) (char \*s, long t, int len)  
*Copy constant string to string for specified number of characters.*
- long [dstrchr](#) (long s, char c)  
*Locate character in constant string.*

### 2.15.1 Detailed Description

This library provides functions for strings located in rom.

Unlike library [cstring.h](#) that uses const char \* pointers, this library uses long for pointers.

Definition in file [dstring.h](#).



## 2.15.2 Function Documentation

### 2.15.2.1 `char* dstreat (char * s, long t)`

The `dstreat()` function appends the `t` string to the `s` string overwriting the ‘`’` character at the end of `s`, and then adds a terminating ‘`’` character. The strings may not overlap, and the `s` string must have enough space for the result.

#### Parameters:

- `s` Address of string in ram.
- `t` Address of string in rom.

#### Returns:

The `dstreat()` function returns a pointer to the destination string `s`.

Definition at line 55 of file `dstring.h`.

References `RomWord`, and `strlen()`.

### 2.15.2.2 `long dstchr (long s, char c)`

The `dstchr()` function searches a constant string for the presence of a specific character.

#### Parameters:

- `s` Address of string in rom.
- `c` Character to locate.

#### Returns:

The `dstchr()` function returns a pointer to the matched character or `NULL` if the character is not found.

Definition at line 80 of file `dstring.h`.

References `RomWord`.

### 2.15.2.3 `int dstcmp (char * s, long t)`

The `dstcmp()` function compares the two strings `s` and `t`.

It returns an integer less than, equal to, or greater than zero if `s` is found, respectively, to be less than, to match, or be greater than `t`.

#### Parameters:

- `s` Address of string in ram.
- `t` Address of string in rom.

#### Returns:

The `dstcmp()` function returns an integer less than, equal to, or greater than zero if `s` is found, respectively, to be less than, to match, or be greater than `t`.

Definition at line 106 of file dstring.h.

References RomWord.

#### 2.15.2.4 char\* dstncpy (char \* s, long t)

The [dstncpy\(\)](#) function copies the string pointed to by t (including the terminating ‘\0’ character) to the array pointed to by s. The strings may not overlap, and the destination string s must be large enough to receive the copy.

##### Parameters:

- s* Address of string in ram.
- t* Address of string in rom.

##### Returns:

The [dstncpy\(\)](#) function returns a pointer to the destination string s.

Definition at line 133 of file dstring.h.

References RomWord.

#### 2.15.2.5 long dstrlen (long s)

The [dstrlen\(\)](#) function calculates the length of the constant string s, not including the terminating ‘\0’ character.

##### Parameters:

- s* Address of string in rom.

##### Returns:

The [dstrlen\(\)](#) function returns the length of constant string s.

Definition at line 29 of file dstring.h.

References RomWord.

#### 2.15.2.6 char\* dstncat (char \* s, long t, int len)

The [dstncat\(\)](#) function appends up to the specified number of characters from the t string to the s string overwriting the ‘\0’ character at the end of s, and then adds a terminating ‘\0’ character.

The strings may not overlap, and the s string must have enough space for the result.

##### Parameters:

- s* Address of string in ram.
- t* Address of string in rom.
- len* Number of characters to copy.

**Returns:**

The `dstrncat()` function returns a pointer to the destination string *s*.

Definition at line 161 of file `dstring.h`.

References `RomWord`, and `strlen()`.

**2.15.2.7 int dstrncmp (char \* s, long t, int len)**

The `dstrncmp()` function compares the two strings *s* and *t* for the first (at most) *n* characters.

It returns an integer less than, equal to, or greater than zero if *s* is (or the first *n* bytes thereof) found, respectively, to be less than, to match, or be greater than *t*.

**Parameters:**

*s* Address of string in ram.

*t* Address of string in rom.

*len* Number of characters to compare.

**Returns:**

The `dstrncmp()` function returns an integer less than, equal to, or greater than zero if *s* (or the first *n* bytes thereof) is found, respectively, to be less than, to match, or be greater than *t*.

Definition at line 193 of file `dstring.h`.

References `RomWord`.

**2.15.2.8 char\* dstrncpy (char \* s, long t, int len)**

The `dstrncpy()` function copies up to the specified number of characters from the *t* string (including the terminating ‘\0’ character) to the array pointed to by *s*. The strings may not overlap, and the destination string *s* must be large enough to receive the copy.

**Parameters:**

*s* Address of string in ram.

*t* Address of string in rom.

*len* Number of bytes to copy.

**Returns:**

The `dstrncpy()` function returns a pointer to the destination string *s*.

Definition at line 223 of file `dstring.h`.

References `RomWord`.

### 2.15.2.9 long dstrchr (long *s*, char *c*)

The [dstrchr\(\)](#) function searches a constant string for the presence of a specific character.

**Parameters:**

- s* Address of string in rom.
- c* Character to locate.

**Returns:**

The [dstrchr\(\)](#) function returns a pointer to the last occurrence of the character *c* in the string *s*.

Definition at line 255 of file dstring.h.

References RomWord.

## 2.16 lib/text/string/string.h File Reference

Library for string support.

```
#include <system/memory.h>
```

### Functions

- char \* [strcat](#) (char \*s, char \*t)  
*Concatenate string to string.*
- char \* [strchr](#) (char \*s, char c)  
*Locate character in string.*
- int [strcmp](#) (char \*s, char \*t)  
*Compare two strings.*
- char \* [strcpy](#) (char \*s, char \*t)  
*Copy string to string.*
- int [strlen](#) (char \*s)  
*Calculate the length of a string.*
- char \* [strncat](#) (char \*s, char \*t, int len)  
*Concatenate string to string for specified number of characters.*
- int [strncmp](#) (char \*s, char \*t, int len)  
*Compare two strings for up to a specified number of bytes.*
- char \* [strncpy](#) (char \*s, char \*t, int len)  
*Copy string to string for specified number of characters.*
- char \* [strrchr](#) (char \*s, char c)  
*Locate character in string.*

### 2.16.1 Detailed Description

This library provides functions for strings located in ram.

Definition in file [string.h](#).

### 2.16.2 Function Documentation

#### 2.16.2.1 char\* strcat (char \* s, char \* t)

The [strcat\(\)](#) function appends the t string to the s string overwriting the ‘\0’ character at the end of s, and then adds a terminating ‘\0’ character. The strings may not overlap, and the s string must have enough space for the result.

**Parameters:**

*s* Address of string in ram.

*t* Address of string in ram.

**Returns:**

The [strcat\(\)](#) function returns a pointer to the destination string *s*.

Definition at line 49 of file string.h.

References [strlen\(\)](#).

**2.16.2.2 char\* strchr (char \* s, char c)**

The [strchr\(\)](#) function searches a string for the presence of a specific character.

**Parameters:**

*s* Address of string in ram.

*c* Character to locate.

**Returns:**

The [strchr\(\)](#) function returns a pointer to the matched character or NULL if the character is not found.

Definition at line 73 of file string.h.

**2.16.2.3 int strcmp (char \* s, char \* t)**

The [strcmp\(\)](#) function compares the two strings *s* and *t*.

It returns an integer less than, equal to, or greater than zero if *s* is found, respectively, to be less than, to match, or be greater than *t*.

**Parameters:**

*s* Address of string in ram.

*t* Address of string in ram.

**Returns:**

The [strcmp\(\)](#) function returns an integer less than, equal to, or greater than zero if *s* is found, respectively, to be less than, to match, or be greater than *t*.

Definition at line 96 of file string.h.

**2.16.2.4 char\* strcpy (char \* s, char \* t)**

The [strcpy\(\)](#) function copies the string pointed to by *t* (including the terminating ‘\0’ character) to the array pointed to by *s*. The strings may not overlap, and the destination string *s* must be large enough to receive the copy.

**Parameters:**

*s* Address of string in ram.

*t* Address of string in ram.

**Returns:**

The [strcpy\(\)](#) function returns a pointer to the destination string *s*.

Definition at line 125 of file string.h.

**2.16.2.5 int strlen (char \* s)**

The [strlen\(\)](#) function calculates the length of the string *s*, not including the terminating ‘\0’ character.

**Parameters:**

*s* Address of string in ram.

**Returns:**

The [strlen\(\)](#) function returns the length of string *s*.

Definition at line 26 of file string.h.

**2.16.2.6 char\* strcat (char \* s, char \* t, int len)**

The [strncat\(\)](#) function appends up to the specified number of characters from the *t* string to the *s* string overwriting the ‘\0’ character at the end of *s*, and then adds a terminating ‘\0’ character.

The strings may not overlap, and the *s* string must have enough space for the result.

**Parameters:**

*s* Address of string in ram.

*t* Address of string in ram.

*len* Number of characters to copy.

**Returns:**

The [strncat\(\)](#) function returns a pointer to the destination string *s*.

Definition at line 153 of file string.h.

References [strlen\(\)](#).

**2.16.2.7 int strncmp (char \* s, char \* t, int len)**

The [strncmp\(\)](#) function compares the two strings *s* and *t* for the first (at most) *n* characters.

It returns an integer less than, equal to, or greater than zero if *s* is (or the first *n* bytes thereof) found, respectively, to be less than, to match, or be greater than *t*.

**Parameters:**

- s* Address of string in ram.
- t* Address of string in ram.
- len* Number of characters to compare.

**Returns:**

The `strncmp()` function returns an integer less than, equal to, or greater than zero if *s* (or the first *n* bytes thereof) is found, respectively, to be less than, to match, or be greater than *t*.

Definition at line 186 of file string.h.

**2.16.2.8 char\* strncpy (char \* s, char \* t, int len)**

The `strncpy()` function copies up to the specified number of characters from the *t* string (including the terminating ‘\0’ character) to the array pointed to by *s*. The strings may not overlap, and the destination string *s* must be large enough to receive the copy.

**Parameters:**

- s* Address of string in ram.
- t* Address of string in ram.
- len* Number of bytes to copy.

**Returns:**

The `strncpy()` function returns a pointer to the destination string *s*.

Definition at line 216 of file string.h.

**2.16.2.9 char\* strrchr (char \* s, char c)**

The `strrchr()` function searches a string for the presence of a specific character.

**Parameters:**

- s* Address of string in ram.
- c* Character to locate.

**Returns:**

The `strrchr()` function returns a pointer to the last occurrence of the character *c* in the string *s*.

Definition at line 251 of file string.h.



## 2.17 lib/virtualperipheral/uart/vpUart.h File Reference

Library for virtual peripheral vpUart.

```
#include <system/memory.h>
```

### Defines

- #define [vpUartRx](#)(name, datapin, dataconfig, hspin, hsconfig, baud, ratio, format)  
*Define a receive uart.*
- #define [vpUartRx\\_ISR](#)  
*Interrupt code for virtual peripheral receive uart.*
- #define [vpUartTx](#)(name, datapin, dataconfig, hspin, hsconfig, baud, ratio, format)  
*Define a transmit uart.*
- #define [vpUartTx\\_ISR](#)  
*Interrupt code for virtual peripheral transmit uart.*
- #define [vpUartTx\\_sendByte](#)(uart, value)  
*Write value to vpUart transmitter.*

### Functions

- char [vpUart\\_queue](#) (char W)  
*Get address of vpUart queue.*
- bit [vpUartRx\\_byteAvailable](#) (char W)  
*Test if vpUart receiver has a byte waiting.*
- void [vpUartRx\\_hsOff](#) (char W)  
*Turn vpUart receiver handshake (RTS) off.*
- void [vpUartRx\\_hsOn](#) (char W)  
*Turn vpUart receiver handshake (RTS) on.*
- bit [vpUartRx\\_parityError](#) (char W)  
*Get vpUart receiver error result.*
- char [vpUartRx\\_receiveByte](#) (char W)  
*Receive byte from vpUart receiver.*
- bit [vpUartTx\\_ready](#) (char W)  
*Test if vpUart transmitter is ready.*

### 2.17.1 Detailed Description

This library provides virtual peripheral uarts for both transmit and receive. These uarts are fully featured: 7/8 databits, none/even/odd parity, invert/normal mode.

Definition in file [vpUart.h](#).

### 2.17.2 Define Documentation

#### 2.17.2.1 #define vpUartRx(name, datapin, dataconfig, hspin, hsconfig, baud, ratio, format)

The macro `vpUartRx()` defines a receive uart. Its parameters are stored in rom.

You install this uart by calling `vph = vpInstall(name)` and you remove the uart by calling `vpUninstall(vph)`.

How to use:

```
vpUartRx(RX1, PORTPIN_RA2+PORTPIN_INV, LOWINPUT_LOWINPUT, 0, 0, 9600, 1, E72)
char rx1 = vpInstall(RX1);
```

The above defines a receive uart with inverted mode for datapin RA.2 that uses 9600 baud for 7 databits with even parity and 2 stopbits. Note the absent ';' after the definition.

#### Parameters:

- name** Name for this uart. The name must be unique as it is used as a label for rom data.
- datapin** Pin to receive data. Use `PORTPIN_XX`, optionally ORed with `PORTPIN_INV` (inverted mode) and/or `PORTPIN_OC` (open collector mode).
- dataconfig** Configuration for the datapin. This specifies the pin states after installation and removal of the uart. Its format is `XX_YY` in which `XX` defines the state after installation and `YY` defines the state after removal of the uart. Possible values for `XX` and `YY` are: `LEAVE`, `LOWOUTPUT`, `HIGHOUTPUT`, `LOWINPUT`, `HIGHINPUT`. Because the datapin is an input pin, only `LOWINPUT` and `HIGHINPUT` make sense.
- hspin** Pin for handshake. Use `PORTPIN_XX`, optionally ORed with `PORTPIN_INV` (inverted mode) and/or `PORTPIN_OC` (open collector mode). Use 0 for no handshake.
- hsconfig** Configuration for the handshake pin. This specifies the pin states after installation and removal of the uart. Its format is `XX_YY` in which `XX` defines the state after installation and `YY` defines the state after removal of the uart. Possible values for `XX` and `YY` are: `LEAVE`, `LOWOUTPUT`, `HIGHOUTPUT`, `LOWINPUT`, `HIGHINPUT`. Because the handshake pin is an output pin, only `LOWOUTPUT` and `HIGHOUTPUT` make sense. Use 0 for no handshake.
- baud** The baudrate for the uart. You can specify the usual baudrates: 1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200. Which baudrates are supported is determined by the `CPUFREQ` and `MAXBAUD` settings.
- ratio** The ratio for the uart. This is to support threaded interrupt code. Not implemented yet. For the moment use value 1.
- format** The format for the uart. This specifies the dataformat. Values available are:
  - E71 even parity, 7 databits, 1 stopbit

- E72 even parity, 7 databits, 2 stopbits
- E81 even parity, 8 databits, 1 stopbit
- O71 odd parity, 7 databits, 1 stopbit
- O72 odd parity, 7 databits, 2 stopbits
- O81 odd parity, 8 databits, 1 stopbit
- N71 no parity, 7 databits, 1 stopbit
- N72 no parity, 7 databits, 2 stopbits
- N81 no parity, 8 databits, 1 stopbit
- N82 no parity, 8 databits, 2 stopbits

Definition at line 77 of file vpUart.h.

### 2.17.2.2 #define vpUartRx\_ISR

This part only receives the serial bits.

Definition at line 245 of file vpUart.h.

### 2.17.2.3 #define vpUartTx(name, datapin, dataconfig, hspin, hsconfig, baud, ratio, format)

The macro `vpUartTx()` defines a transmit uart. Its parameters are stored in rom.

You install this uart by calling `vph = vpInstall(name)` and you remove the uart by calling `vpUninstall(vph)`.

How to use:

```
vpUartTx(TX1, PORTPIN_RA1, HIGHOUTPUT_HIGHOUTPUT, 0, 0, 9600, 1, E72)
char tx1 = vpInstall(TX1);
```

The above defines a transmit uart with normal mode for datapin RA.1 that uses 9600 baud for 7 databits with even parity and 2 stopbits. Note the absent `';` after the definition.

#### Parameters:

**name** Name for this uart. The name must be unique as it is used as a label for rom data.

**datapin** Pin to transmit data. Use `PORTPIN_XX`, optionally ORed with `PORTPIN_INV` (inverted mode) and/or `PORTPIN_OC` (open collector mode).

**dataconfig** Configuration for the datapin. This specifies the pin states after installation and removal of the uart. Its format is `XX_YY` in which `XX` defines the state after installation and `YY` defines the state after removal of the uart. Possible values for `XX` and `YY` are: `LEAVE`, `LOWOUTPUT`, `HIGHOUTPUT`, `LOWINPUT`, `HIGHINPUT`. Because the datapin is an output pin, only `LOWOUTPUT` and `HIGHOUTPUT` make sense.

**hspin** Pin for handshake. Use `PORTPIN_XX`, optionally ORed with `PORTPIN_INV` (inverted mode) and/or `PORTPIN_OC` (open collector mode). Use 0 for no handshake.

**hsconfig** Configuration for the handshake pin. This specifies the pin states after installation and removal of the uart. Its format is `XX_YY` in which `XX` defines the state after installation and `YY` defines the state after removal of the uart. Possible values for `XX` and `YY` are: `LEAVE`, `LOWOUTPUT`, `HIGHOUTPUT`, `LOWINPUT`, `HIGHINPUT`. Because the handshake pin is an input pin, only `LOWINPUT` and `HIGHINPUT` make sense. Use 0 for no handshake.

**baud** The baudrate for the uart. You can specify the usual baudrates: 1200, 2400, 4800, 9600, 19200, 28800, 38400, 57600, 115200. Which baudrates are supported is determined by the CPUFREQ and MAXBAUD settings.

**ratio** The ratio for the uart. This is to support threaded interrupt code. Not implemented yet. For the moment use value 1.

**format** The format for the uart. This specifies the dataformat. Values available are:

- E71 even parity, 7 databits, 1 stopbit
- E72 even parity, 7 databits, 2 stopbits
- E81 even parity, 8 databits, 1 stopbit
- O71 odd parity, 7 databits, 1 stopbit
- O72 odd parity, 7 databits, 2 stopbits
- O81 odd parity, 8 databits, 1 stopbit
- N71 no parity, 7 databits, 1 stopbit
- N72 no parity, 7 databits, 2 stopbits
- N81 no parity, 8 databits, 1 stopbit
- N82 no parity, 8 databits, 2 stopbits

Definition at line 174 of file vpUart.h.

#### 2.17.2.4 #define vpUartTx\_ISR

This part only transmits the serial bits.

Definition at line 359 of file vpUart.h.

#### 2.17.2.5 #define vpUartTx\_sendByte(uart, value)

The function vpUart\_sendByte() sends a byte to the vpUart transmitter.

It must be tested whether the transmitter is ready to accept a new byte.

How to use:

```
if (vpUartTx_ready(tx)) {
    _MainTemp = value; //value to write must be in a global ram location
    vpUartTx_sendByte(tx, _MainTemp);
}
```

#### Parameters:

**uart** vpUart address.

**value** Value to write. The value MUST have been stored in a global ram location (0x00-0x0F).

Definition at line 635 of file vpUart.h.

### 2.17.3 Function Documentation

#### 2.17.3.1 char vpUart\_queue (char W)

The function vpUart\_queue() returns the address of the char array that is included

in the vpUart dataset. This array is initialized as a queue, you may however use it as you see fit. Just make sure to get its size before using it for some other purpose. The reason is that the array has some queue properties that may be overwritten when not used as a queue.

**Parameters:**

*W* vpUart address.

**Returns:**

Address of queue.

Definition at line 709 of file vpUart.h.

**2.17.3.2 bit vpUartRx\_byteAvailable (char *W*)**

This function checks if a byte is available.

**Parameters:**

*W* vpUart address.

**Returns:**

True if byte available.

Definition at line 462 of file vpUart.h.

**2.17.3.3 void vpUartRx\_hsOff (char *W*)**

Post-isr code for virtual peripheral receive uart.

This function turns the optional handshake output pin (RTS) off (remote sender may not transmit).

**Parameters:**

*W* vpUart address.

Definition at line 585 of file vpUart.h.

**2.17.3.4 void vpUartRx\_hsOn (char *W*)**

Post-isr code for virtual peripheral receive uart.

This function turns the optional handshake output pin (RTS) on (remote sender may transmit).

**Parameters:**

*W* vpUart address.

Definition at line 565 of file vpUart.h.

### 2.17.3.5 bit vpUartRx\_parityError (char W)

Post-isr code for virtual peripheral receive uart.

This function returns the errorbit that is updated by [vpUartRx\\_receiveByte\(\)](#) so it should be called after [vpUartRx\\_receiveByte\(\)](#) is called (only when parity is used).

#### Parameters:

W vpUart address.

#### Returns:

errorbit updated by [vpUartRx\\_receiveByte\(\)](#).

Definition at line 546 of file vpUart.h.

### 2.17.3.6 char vpUartRx\_receiveByte (char W)

Post-isr code for virtual peripheral receive uart.

This function grabs a received byte and checks for parity error if required.

How to use:

```
if (vpUartRx_byteAvailable(rx)) {  
    value = vpUartRx_receiveByte(rx);  
}
```

#### Parameters:

W vpUart address.

#### Returns:

received byte.

Definition at line 488 of file vpUart.h.

### 2.17.3.7 bit vpUartTx\_ready (char W)

Pre-isr code for virtual peripheral transmit uart.

This part checks if the transmitter is idle and the optional handshake input pin (CTS) allows sending.

#### Parameters:

W vpUart address.

#### Returns:

True if CTS is on or no handshake is used AND the transmitter is idle (transmitting allowed).

Definition at line 606 of file vpUart.h.

## 2.18 lib/virtualperipheral/vplib.h File Reference

Library for virtual peripheral support.

### Defines

- #define [VPBANK0](#) (VPRAM+0x00)
- #define [VPLIST](#) (VPRAM + (VPSLOTS<<4))

### Functions

- void [vpInit](#) (void)  
*Initialize VP environment.*
- char [vpInstall](#) (long addr)  
*Install virtual peripheral.*
- void [vpUninstall](#) (char bnk)  
*Uninstall virtual peripheral.*

### 2.18.1 Detailed Description

This library provides a generic interface to virtual peripherals. Basic elements of a virtual peripheral are a set of data and functions that access that data. Some VP's have interrupt code, or mainline code or both. One property of this generic interface is that the VP dataset can be located in any of the rambanks assigned to VP's. This makes it possible to use dynamic loading of VP's. This means VP's can be installed and uninstalled as needed.

Definition in file [vplib.h](#).

### 2.18.2 Define Documentation

#### 2.18.2.1 #define VPBANK0 (VPRAM+0x00)

Definition at line 81 of file [vplib.h](#).

#### 2.18.2.2 #define VPLIST (VPRAM + (VPSLOTS<<4))

Definition at line 45 of file [vplib.h](#).

### 2.18.3 Function Documentation

#### 2.18.3.1 void vpInit (void)

The function [vpInit\(\)](#) initializes the [vpList](#).

It must be called prior to any other function from this library.

Definition at line 125 of file vplib.h.

### 2.18.3.2 char vpInstall (long *addr*)

The function `vpInstall()` installs a VP into the first free entry of the vpList.

If all vp slots are occupied, this function returns 0.

How to use:

```
char vph; //handle for vp
vph = vpInstall(MYVP);
//use the VP for some time
//until no longer needed
vpUninstall(vph);
```

#### Parameters:

*addr* Address of virtual peripheral initialization data.

#### Returns:

handle for installed virtual peripheral if succesful, or 0 if not succesful.

The format of the handle:

- high nibble (b7-b4) are bits b7-b4 of rambank where virtual peripheral is installed
- low nibble (b3-b0) are the index in the vpListType array

Definition at line 200 of file vplib.h.

References RomCopy(), and RomWord.

### 2.18.3.3 void vpUninstall (char *bnk*)

The function `vpUninstall()` removes an installed VP from the vpList.

The handle must be considered invalid after using this function.

A new VP can be installed by calling `vpInstall()`.

How to use:

```
char vph; //handle for vp
vph = vpInstall(MYVP);
//use the VP for some time
//until no longer needed
vpUninstall(vph);
```

#### Parameters:

*bnk* Handle of installed VP.

Definition at line 248 of file vplib.h.



# Index

- `_CHIP_CODESIZE_`
    - `SX18.H`, 15
    - `SX20.H`, 18
    - `SX28.H`, 21
    - `SX48.H`, 24
    - `SX52.H`, 27
  - `_CHIP_SX18_`
    - `SX18.H`, 15
  - `_CHIP_SX20_`
    - `SX20.H`, 18
  - `_CHIP_SX28_`
    - `SX28.H`, 21
  - `_CHIP_SX48_`
    - `SX48.H`, 24
  - `_CHIP_SX52_`
    - `SX52.H`, 27
  - `_SystemRamAddress`
    - `memory.h`, 44
  - `_SystemRamIndex`
    - `memory.h`, 45
- `abs`
  - `stdlib.h`, 74
- `atoi`
  - `stdlib.h`, 74
- `atoiB`
  - `stdlib.h`, 74
- `Baud_divide`
  - `defines.h`, 31
- `cstring`
  - `cstring.h`, 81
- `cstringr`
  - `cstring.h`, 81
- `strcmp`
  - `cstring.h`, 81
- `strcpy`
  - `cstring.h`, 81
- `cstring.h`
  - `cstringcat`, 81
  - `cstringchr`, 81
  - `strcmp`, 81
  - `strcpy`, 81
  - `strlen`, 82
  - `cstringcat`, 82
  - `strcmp`, 82
  - `strcpy`, 83
  - `cstringchr`, 83
- `strlen`
  - `cstring.h`, 82
- `cstringcat`
  - `cstring.h`, 82
- `strcmp`
  - `cstring.h`, 82
- `strcpy`
  - `cstring.h`, 83
- `cstringchr`
  - `cstring.h`, 83
- `ctype.h`
  - `isalnum`, 69
  - `isalpha`, 69
  - `isascii`, 69
  - `isctrl`, 69
  - `isdigit`, 70
  - `isgraph`, 70
  - `islower`, 70
  - `isprint`, 70
  - `ispunct`, 71
  - `isspace`, 71
  - `isupper`, 71
  - `isxdigit`, 71
  - `toascii`, 72
  - `tolower`, 72
  - `toupper`, 72
- `DDR_init`
  - `memory.h`, 45
- `DDR_update`
  - `portpin.h`, 57
- `defines.h`
  - `Baud_divide`, 31
  - `E71`, 31
  - `E72`, 31
  - `E81`, 31
  - `HIGHINPUT_HIGHINPUT`, 31
  - `HIGHINPUT_HIGHOUTPUT`, 31
  - `HIGHINPUT_LEAVE`, 32
  - `HIGHINPUT_LOWINPUT`, 32
  - `HIGHINPUT_LOWOUTPUT`, 32

HIGHOUTPUT\_HIGHINPUT, 32  
 HIGHOUTPUT\_HIGHOUTPUT, 32  
 HIGHOUTPUT\_LEAVE, 32  
 HIGHOUTPUT\_LOWINPUT, 32  
 HIGHOUTPUT\_LOWOUTPUT, 32  
 INTPERIOD, 32  
 LEAVE\_HIGHINPUT, 32  
 LEAVE\_HIGHOUTPUT, 32  
 LEAVE\_LEAVE, 33  
 LEAVE\_LOWINPUT, 33  
 LEAVE\_LOWOUTPUT, 33  
 LOWINPUT\_HIGHINPUT, 33  
 LOWINPUT\_HIGHOUTPUT, 33  
 LOWINPUT\_LEAVE, 33  
 LOWINPUT\_LOWINPUT, 33  
 LOWINPUT\_LOWOUTPUT, 33  
 LOWOUTPUT\_HIGHOUTPUT, 33  
 LOWOUTPUT\_HIGHINPUT, 33  
 LOWOUTPUT\_LEAVE, 33  
 LOWOUTPUT\_LOWINPUT, 34  
 LOWOUTPUT\_LOWOUTPUT, 34  
 MAXBAUD, 34  
 MSCOUNT, 34  
 N71, 34  
 N72, 34  
 N81, 34  
 N82, 34  
 O71, 34  
 O72, 34  
 O81, 34  
 PORT\_RA, 35  
 PORT\_RB, 35  
 PORT\_RC, 35  
 PORT\_RD, 35  
 PORT\_RE, 35  
 PORTPIN\_INV, 35  
 PORTPIN\_OC, 35  
 PORTPIN\_RA, 35  
 PORTPIN\_RA0, 35  
 PORTPIN\_RA1, 35  
 PORTPIN\_RA2, 35  
 PORTPIN\_RA3, 36  
 PORTPIN\_RA4, 36  
 PORTPIN\_RA5, 36  
 PORTPIN\_RA6, 36  
 PORTPIN\_RA7, 36  
 PORTPIN\_RB, 36  
 PORTPIN\_RB0, 36  
 PORTPIN\_RB1, 36  
 PORTPIN\_RB2, 36  
 PORTPIN\_RB3, 36  
 PORTPIN\_RB4, 36  
 PORTPIN\_RB5, 37  
 PORTPIN\_RB6, 37  
 PORTPIN\_RB7, 37  
 PORTPIN\_RC, 37  
 PORTPIN\_RC0, 37  
 PORTPIN\_RC1, 37  
 PORTPIN\_RC2, 37  
 PORTPIN\_RC3, 37  
 PORTPIN\_RC4, 37  
 PORTPIN\_RC5, 37  
 PORTPIN\_RC6, 37  
 PORTPIN\_RC7, 38  
 PORTPIN\_RD, 38  
 PORTPIN\_RD0, 38  
 PORTPIN\_RD1, 38  
 PORTPIN\_RD2, 38  
 PORTPIN\_RD3, 38  
 PORTPIN\_RD4, 38  
 PORTPIN\_RD5, 38  
 PORTPIN\_RD6, 38  
 PORTPIN\_RD7, 38  
 PORTPIN\_RE, 38  
 PORTPIN\_RE0, 39  
 PORTPIN\_RE1, 39  
 PORTPIN\_RE2, 39  
 PORTPIN\_RE3, 39  
 PORTPIN\_RE4, 39  
 PORTPIN\_RE5, 39  
 PORTPIN\_RE6, 39  
 PORTPIN\_RE7, 39  
 PS\_000, 39  
 PS\_001, 39  
 PS\_010, 39  
 PS\_011, 40  
 PS\_100, 40  
 PS\_101, 40  
 PS\_110, 40  
 PS\_111, 40  
 RTCC\_FE, 40  
 RTCC\_ID, 40  
 RTCC\_INC\_EXT, 40  
 RTCC\_ON, 40  
 RTCC\_PS\_OFF, 40  
 RTCC\_PS\_ON, 40  
 uartfs, 41  
 USCOUNT, 41  
 VP\_TIMER16, 41  
 VP\_UARTRX, 41  
 VP\_UARTTX, 41  
 VP\_UNINSTALLED, 41  
 dstrcat  
     dstring.h, 85  
 dstrchr  
     dstring.h, 85  
 dstncmp  
     dstring.h, 85

- dstrepy
  - dstring.h, 86
- dstring.h
  - dstreat, 85
  - dstrechr, 85
  - dstremp, 85
  - dstrepy, 86
  - dstrlen, 86
  - dstrencat, 86
  - dstrencmp, 87
  - dstrencpy, 87
  - dstrrchr, 87
- dstrlen
  - dstring.h, 86
- dstrencat
  - dstring.h, 86
- dstrencmp
  - dstring.h, 87
- dstrencpy
  - dstring.h, 87
- dstrrchr
  - dstring.h, 87
- dtoi
  - stdlib.h, 75
- E71
  - defines.h, 31
- E72
  - defines.h, 31
- E81
  - defines.h, 31
- HIGHINPUT\_HIGHINPUT
  - defines.h, 31
- HIGHINPUT\_HIGHOUTPUT
  - defines.h, 31
- HIGHINPUT\_LEAVE
  - defines.h, 32
- HIGHINPUT\_LOWINPUT
  - defines.h, 32
- HIGHINPUT\_LOWOUTPUT
  - defines.h, 32
- HIGHOUTPUT\_HIGHINPUT
  - defines.h, 32
- HIGHOUTPUT\_HIGHOUTPUT
  - defines.h, 32
- HIGHOUTPUT\_LEAVE
  - defines.h, 32
- HIGHOUTPUT\_LOWINPUT
  - defines.h, 32
- HIGHOUTPUT\_LOWOUTPUT
  - defines.h, 32
- INTPERIOD
  - defines.h, 32
- isalnum
  - ctype.h, 69
- isalpha
  - ctype.h, 69
- isascii
  - ctype.h, 69
- iscentrl
  - ctype.h, 69
- isdigit
  - ctype.h, 70
- isgraph
  - ctype.h, 70
- islower
  - ctype.h, 70
- isprint
  - ctype.h, 70
- ispunct
  - ctype.h, 71
- isspace
  - ctype.h, 71
- isupper
  - ctype.h, 71
- isxdigit
  - ctype.h, 71
- itoa
  - stdlib.h, 75
- itoab
  - stdlib.h, 75
- itod
  - stdlib.h, 76
- itoo
  - stdlib.h, 76
- itou
  - stdlib.h, 76
- itox
  - stdlib.h, 77
- LargeArrayAddress
  - memory.h, 46
- LargeArrayIndex
  - memory.h, 46
- LargeArrayRead
  - memory.h, 43
- LargeArrayWrite
  - memory.h, 43
- LEAVE\_HIGHINPUT
  - defines.h, 32
- LEAVE\_HIGHOUTPUT
  - defines.h, 32
- LEAVE\_LEAVE
  - defines.h, 33
- LEAVE\_LOWINPUT
  - defines.h, 33

LEAVE\_LOWOUTPUT  
     defines.h, 33  
 left  
     stdlib.h, 77  
 lib/datastructures/queue/objQueue.h, 3  
 lib/datastructures/stack/objStack.h, 9  
 lib/sxdevice/SX18.H, 14  
 lib/sxdevice/SX20.H, 17  
 lib/sxdevice/SX28.H, 20  
 lib/sxdevice/SX48.H, 23  
 lib/sxdevice/SX52.H, 26  
 lib/system/defines.h, 29  
 lib/system/memory.h, 42  
 lib/system/portpin.h, 49  
 lib/taskswitching/Task.h, 59  
 lib/text/character/ctype.h, 68  
 lib/text/conversion/stdlib.h, 73  
 lib/text/string/cstring.h, 80  
 lib/text/string/dstring.h, 84  
 lib/text/string/string.h, 89  
 lib/virtualperipheral/uart/vpUart.h, 93  
 lib/virtualperipheral/vplib.h, 99  
 LOWINPUT\_HIGHINPUT  
     defines.h, 33  
 LOWINPUT\_HIGHOUTPUT  
     defines.h, 33  
 LOWINPUT\_LEAVE  
     defines.h, 33  
 LOWINPUT\_LOWINPUT  
     defines.h, 33  
 LOWINPUT\_LOWOUTPUT  
     defines.h, 33  
 LOWOUTPUT\_HIGHOUTPUT  
     defines.h, 33  
 LOWOUTPUT\_HIGHINPUT  
     defines.h, 33  
 LOWOUTPUT\_LEAVE  
     defines.h, 33  
 LOWOUTPUT\_LOWINPUT  
     defines.h, 34  
 LOWOUTPUT\_LOWOUTPUT  
     defines.h, 34  
 MAXBAUD  
     defines.h, 34  
 membank0  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 24  
     SX52.H, 27  
 membank1  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 24  
     SX52.H, 27  
 membank10  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 24  
     SX52.H, 27  
 membank11  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 24  
     SX52.H, 27  
 membank12  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 24  
     SX52.H, 27  
 membank13  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 24  
     SX52.H, 27  
 membank14  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 24  
     SX52.H, 27  
 membank15  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 25  
     SX52.H, 28  
 membank16  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 25  
     SX52.H, 28  
 membank2  
     SX18.H, 15  
     SX20.H, 18  
     SX28.H, 21  
     SX48.H, 25  
     SX52.H, 28  
 membank3  
     SX18.H, 16  
     SX20.H, 19

- SX28.H, [22](#)
- SX48.H, [25](#)
- SX52.H, [28](#)
- membank4
  - SX18.H, [16](#)
  - SX20.H, [19](#)
  - SX28.H, [22](#)
  - SX48.H, [25](#)
  - SX52.H, [28](#)
- membank5
  - SX18.H, [16](#)
  - SX20.H, [19](#)
  - SX28.H, [22](#)
  - SX48.H, [25](#)
  - SX52.H, [28](#)
- membank6
  - SX18.H, [16](#)
  - SX20.H, [19](#)
  - SX28.H, [22](#)
  - SX48.H, [25](#)
  - SX52.H, [28](#)
- membank7
  - SX18.H, [16](#)
  - SX20.H, [19](#)
  - SX28.H, [22](#)
  - SX48.H, [25](#)
  - SX52.H, [28](#)
- membank8
  - SX18.H, [16](#)
  - SX20.H, [19](#)
  - SX28.H, [22](#)
  - SX48.H, [25](#)
  - SX52.H, [28](#)
- membank9
  - SX18.H, [16](#)
  - SX20.H, [19](#)
  - SX28.H, [22](#)
  - SX48.H, [25](#)
  - SX52.H, [28](#)
- memory.h
  - \_SystemRamAddress, [44](#)
  - \_SystemRamIndex, [45](#)
  - DDR\_init, [45](#)
  - LargeArrayAddress, [46](#)
  - LargeArrayIndex, [46](#)
  - LargeArrayRead, [43](#)
  - LargeArrayWrite, [43](#)
  - RomCopy, [46](#)
  - RomWord, [43](#)
  - x00, [47](#)
  - x0C, [47](#)
  - x0D, [47](#)
  - x0E, [47](#)
  - xF5, [47](#)
  - xF6, [47](#)
  - xF7, [47](#)
  - xF8, [48](#)
  - xF9, [48](#)
- MSCOUNT
  - defines.h, [34](#)
- N71
  - defines.h, [34](#)
- N72
  - defines.h, [34](#)
- N81
  - defines.h, [34](#)
- N82
  - defines.h, [34](#)
- O71
  - defines.h, [34](#)
- O72
  - defines.h, [34](#)
- O81
  - defines.h, [34](#)
- objQueue.h
  - objQueue\_capacity, [5](#)
  - objQueue\_clear, [5](#)
  - objQueue\_dequeue, [6](#)
  - objQueue\_enqueue, [4](#)
  - objQueue\_free, [6](#)
  - objQueue\_init, [5](#)
  - objQueue\_isEmpty, [6](#)
  - objQueue\_isFull, [7](#)
  - objQueue\_length, [7](#)
  - objQueue\_size, [8](#)
- objQueue\_capacity
  - objQueue.h, [5](#)
- objQueue\_clear
  - objQueue.h, [5](#)
- objQueue\_dequeue
  - objQueue.h, [6](#)
- objQueue\_enqueue
  - objQueue.h, [4](#)
- objQueue\_free
  - objQueue.h, [6](#)
- objQueue\_init
  - objQueue.h, [5](#)
- objQueue\_isEmpty
  - objQueue.h, [6](#)
- objQueue\_isFull
  - objQueue.h, [7](#)
- objQueue\_length
  - objQueue.h, [7](#)
- objQueue\_size
  - objQueue.h, [8](#)
- objStack.h

- objStack\_capacity, 11
- objStack\_clear, 11
- objStack\_free, 11
- objStack\_init, 10
- objStack\_isEmpty, 12
- objStack\_isFull, 12
- objStack\_length, 12
- objStack\_pop, 13
- objStack\_push, 10
- objStack\_size, 13
- objStack\_capacity
  - objStack.h, 11
- objStack\_clear
  - objStack.h, 11
- objStack\_free
  - objStack.h, 11
- objStack\_init
  - objStack.h, 10
- objStack\_isEmpty
  - objStack.h, 12
- objStack\_isFull
  - objStack.h, 12
- objStack\_length
  - objStack.h, 12
- objStack\_pop
  - objStack.h, 13
- objStack\_push
  - objStack.h, 10
- objStack\_size
  - objStack.h, 13
- otoi
  - stdlib.h, 77
- pad
  - stdlib.h, 78
- PORT\_RA
  - defines.h, 35
- PORT\_RB
  - defines.h, 35
- PORT\_RC
  - defines.h, 35
- PORT\_RD
  - defines.h, 35
- PORT\_RE
  - defines.h, 35
- portpin.h
  - DDR\_update, 57
  - readPin, 50
  - readPinDirection, 51
  - readPort, 57
  - readPortDirection, 58
  - setPinHigh, 51
  - setPinInput, 52
  - setPinLow, 52
  - setPinOutput, 52
  - setPortLevel, 53
  - setPortPullup, 53
  - setPortSchmittTrigger, 54
  - togglePin, 54
  - togglePinDirection, 55
  - writePinDirection, 55
  - writePinLatch, 56
  - writePortDirection, 56
  - writePortLatch, 56
- PORTPIN\_INV
  - defines.h, 35
- PORTPIN\_OC
  - defines.h, 35
- PORTPIN\_RA
  - defines.h, 35
- PORTPIN\_RA0
  - defines.h, 35
- PORTPIN\_RA1
  - defines.h, 35
- PORTPIN\_RA2
  - defines.h, 35
- PORTPIN\_RA3
  - defines.h, 36
- PORTPIN\_RA4
  - defines.h, 36
- PORTPIN\_RA5
  - defines.h, 36
- PORTPIN\_RA6
  - defines.h, 36
- PORTPIN\_RA7
  - defines.h, 36
- PORTPIN\_RB
  - defines.h, 36
- PORTPIN\_RB0
  - defines.h, 36
- PORTPIN\_RB1
  - defines.h, 36
- PORTPIN\_RB2
  - defines.h, 36
- PORTPIN\_RB3
  - defines.h, 36
- PORTPIN\_RB4
  - defines.h, 36
- PORTPIN\_RB5
  - defines.h, 37
- PORTPIN\_RB6
  - defines.h, 37
- PORTPIN\_RB7
  - defines.h, 37
- PORTPIN\_RC
  - defines.h, 37
- PORTPIN\_RC0
  - defines.h, 37

- PORTPIN\_RC1
  - defines.h, 37
- PORTPIN\_RC2
  - defines.h, 37
- PORTPIN\_RC3
  - defines.h, 37
- PORTPIN\_RC4
  - defines.h, 37
- PORTPIN\_RC5
  - defines.h, 37
- PORTPIN\_RC6
  - defines.h, 37
- PORTPIN\_RC7
  - defines.h, 38
- PORTPIN\_RD
  - defines.h, 38
- PORTPIN\_RD0
  - defines.h, 38
- PORTPIN\_RD1
  - defines.h, 38
- PORTPIN\_RD2
  - defines.h, 38
- PORTPIN\_RD3
  - defines.h, 38
- PORTPIN\_RD4
  - defines.h, 38
- PORTPIN\_RD5
  - defines.h, 38
- PORTPIN\_RD6
  - defines.h, 38
- PORTPIN\_RD7
  - defines.h, 38
- PORTPIN\_RE
  - defines.h, 38
- PORTPIN\_RE0
  - defines.h, 39
- PORTPIN\_RE1
  - defines.h, 39
- PORTPIN\_RE2
  - defines.h, 39
- PORTPIN\_RE3
  - defines.h, 39
- PORTPIN\_RE4
  - defines.h, 39
- PORTPIN\_RE5
  - defines.h, 39
- PORTPIN\_RE6
  - defines.h, 39
- PORTPIN\_RE7
  - defines.h, 39
- PS\_000
  - defines.h, 39
- PS\_001
  - defines.h, 39
- PS\_010
  - defines.h, 39
- PS\_011
  - defines.h, 40
- PS\_100
  - defines.h, 40
- PS\_101
  - defines.h, 40
- PS\_110
  - defines.h, 40
- PS\_111
  - defines.h, 40
- readPin
  - portpin.h, 50
- readPinDirection
  - portpin.h, 51
- readPort
  - portpin.h, 57
- readPortDirection
  - portpin.h, 58
- reverse
  - stdlib.h, 78
- RomCopy
  - memory.h, 46
- RomWord
  - memory.h, 43
- RTCC\_FE
  - defines.h, 40
- RTCC\_ID
  - defines.h, 40
- RTCC\_INC\_EXT
  - defines.h, 40
- RTCC\_ON
  - defines.h, 40
- RTCC\_PS\_OFF
  - defines.h, 40
- RTCC\_PS\_ON
  - defines.h, 40
- setPinHigh
  - portpin.h, 51
- setPinInput
  - portpin.h, 52
- setPinLow
  - portpin.h, 52
- setPinOutput
  - portpin.h, 52
- setPortLevel
  - portpin.h, 53
- setPortPullup
  - portpin.h, 53
- setPortSchmittTrigger
  - portpin.h, 54

- sign
  - stdlib.h, 78
- stdlib.h
  - abs, 74
  - atoi, 74
  - atoib, 74
  - dtoi, 75
  - itoa, 75
  - itoab, 75
  - itod, 76
  - itoo, 76
  - itou, 76
  - itox, 77
  - left, 77
  - otoi, 77
  - pad, 78
  - reverse, 78
  - sign, 78
  - utoi, 79
  - xtoi, 79
- strcat
  - string.h, 89
- strchr
  - string.h, 90
- strcmp
  - string.h, 90
- strcpy
  - string.h, 90
- string.h
  - strcat, 89
  - strchr, 90
  - strcmp, 90
  - strcpy, 90
  - strlen, 91
  - strncat, 91
  - strncmp, 91
  - strncpy, 92
  - strchr, 92
- strlen
  - string.h, 91
- strncat
  - string.h, 91
- strncmp
  - string.h, 91
- strncpy
  - string.h, 92
- strchr
  - string.h, 92
- SX18.H
  - \_CHIP\_CODESIZE\_, 15
  - \_CHIP\_SX18\_, 15
  - membank0, 15
  - membank1, 15
  - membank10, 15
  - membank11, 15
  - membank12, 15
  - membank13, 15
  - membank14, 15
  - membank15, 15
  - membank16, 15
  - membank2, 15
  - membank3, 16
  - membank4, 16
  - membank5, 16
  - membank6, 16
  - membank7, 16
  - membank8, 16
  - membank9, 16
- SX20.H
  - \_CHIP\_CODESIZE\_, 18
  - \_CHIP\_SX20\_, 18
  - membank0, 18
  - membank1, 18
  - membank10, 18
  - membank11, 18
  - membank12, 18
  - membank13, 18
  - membank14, 18
  - membank15, 18
  - membank16, 18
  - membank2, 18
  - membank3, 19
  - membank4, 19
  - membank5, 19
  - membank6, 19
  - membank7, 19
  - membank8, 19
  - membank9, 19
- SX28.H
  - \_CHIP\_CODESIZE\_, 21
  - \_CHIP\_SX28\_, 21
  - membank0, 21
  - membank1, 21
  - membank10, 21
  - membank11, 21
  - membank12, 21
  - membank13, 21
  - membank14, 21
  - membank15, 21
  - membank16, 21
  - membank2, 21
  - membank3, 22
  - membank4, 22
  - membank5, 22
  - membank6, 22
  - membank7, 22
  - membank8, 22
  - membank9, 22



- SX48.H
  - [\\_CHIP\\_CODESIZE\\_](#), 24
  - [\\_CHIP\\_SX48\\_](#), 24
  - [membank0](#), 24
  - [membank1](#), 24
  - [membank10](#), 24
  - [membank11](#), 24
  - [membank12](#), 24
  - [membank13](#), 24
  - [membank14](#), 24
  - [membank15](#), 25
  - [membank16](#), 25
  - [membank2](#), 25
  - [membank3](#), 25
  - [membank4](#), 25
  - [membank5](#), 25
  - [membank6](#), 25
  - [membank7](#), 25
  - [membank8](#), 25
  - [membank9](#), 25
- SX52.H
  - [\\_CHIP\\_CODESIZE\\_](#), 27
  - [\\_CHIP\\_SX52\\_](#), 27
  - [membank0](#), 27
  - [membank1](#), 27
  - [membank10](#), 27
  - [membank11](#), 27
  - [membank12](#), 27
  - [membank13](#), 27
  - [membank14](#), 27
  - [membank15](#), 28
  - [membank16](#), 28
  - [membank2](#), 28
  - [membank3](#), 28
  - [membank4](#), 28
  - [membank5](#), 28
  - [membank6](#), 28
  - [membank7](#), 28
  - [membank8](#), 28
  - [membank9](#), 28
- TASK
  - [Task.h](#), 61
- Task.h
  - [TASK](#), 61
  - [Task\\_ISR](#), 61
  - [TaskDisable](#), 63
  - [TaskEnable](#), 63
  - [TaskInit](#), 63
  - [TASKINTERVAL](#), 62
  - [TASKKERNEL](#), 62
  - [TaskReschedule](#), 63
  - [TaskRomCopy](#), 64
  - [TASKRUNONCE](#), 62
  - [TaskSchedule](#), 64
  - [TaskSet](#), 62
  - [TaskSlot](#), 65
  - [TaskSlotAvailable](#), 65
  - [TaskSlotStart](#), 65
  - [TaskSlotStop](#), 66
  - [TASKSTART](#), 62
  - [TaskStart](#), 66
  - [TaskStop](#), 66
  - [TaskSwitch](#), 66
  - [TaskTimer](#), 67
  - [toascii](#), 72
  - [togglePin](#)
- [TaskSchedule](#), 64
- [TaskSet](#), 62
- [TaskSlot](#), 65
- [TaskSlotAvailable](#), 65
- [TaskSlotStart](#), 65
- [TaskSlotStop](#), 66
- [TASKSTART](#), 62
- [TaskStart](#), 66
- [TaskStop](#), 66
- [TaskSwitch](#), 66
- [TaskTimer](#), 67
- [Task\\_ISR](#)
  - [Task.h](#), 61
- [TaskDisable](#)
  - [Task.h](#), 63
- [TaskEnable](#)
  - [Task.h](#), 63
- [TaskInit](#)
  - [Task.h](#), 63
- [TASKINTERVAL](#)
  - [Task.h](#), 62
- [TASKKERNEL](#)
  - [Task.h](#), 62
- [TaskReschedule](#)
  - [Task.h](#), 63
- [TaskRomCopy](#)
  - [Task.h](#), 64
- [TASKRUNONCE](#)
  - [Task.h](#), 62
- [TaskSchedule](#)
  - [Task.h](#), 64
- [TaskSet](#)
  - [Task.h](#), 62
- [TaskSlot](#)
  - [Task.h](#), 65
- [TaskSlotAvailable](#)
  - [Task.h](#), 65
- [TaskSlotStart](#)
  - [Task.h](#), 65
- [TaskSlotStop](#)
  - [Task.h](#), 66
- [TASKSTART](#)
  - [Task.h](#), 62
- [TaskStart](#)
  - [Task.h](#), 66
- [TaskStop](#)
  - [Task.h](#), 66
- [TaskSwitch](#)
  - [Task.h](#), 66
- [TaskTimer](#)
  - [Task.h](#), 67
- [toascii](#)
  - [ctype.h](#), 72
- [togglePin](#)

- portpin.h, 54
- togglePinDirection
  - portpin.h, 55
- tolower
  - ctype.h, 72
- toupper
  - ctype.h, 72
- uartfs
  - defines.h, 41
- USCOUNT
  - defines.h, 41
- atoi
  - stdlib.h, 79
- VP\_TIMER16
  - defines.h, 41
- VP\_UARTRX
  - defines.h, 41
- VP\_UARTTX
  - defines.h, 41
- VP\_UNINSTALLED
  - defines.h, 41
- VPBANK0
  - vplib.h, 99
- vpInit
  - vplib.h, 99
- vpInstall
  - vplib.h, 100
- vplib.h
  - VPBANK0, 99
  - vpInit, 99
  - vpInstall, 100
  - VPLIST, 99
  - vpUninstall, 100
- VPLIST
  - vplib.h, 99
- vpUart.h
  - vpUart\_queue, 96
  - vpUartRx, 94
  - vpUartRx\_byteAvailable, 97
  - vpUartRx\_hsOff, 97
  - vpUartRx\_hsOn, 97
  - vpUartRx\_ISR, 95
  - vpUartRx\_parityError, 97
  - vpUartRx\_receiveByte, 98
  - vpUartTx, 95
  - vpUartTx\_ISR, 96
  - vpUartTx\_ready, 98
  - vpUartTx\_sendByte, 96
- vpUart\_queue
  - vpUart.h, 96
- vpUartRx
  - vpUart.h, 94
- vpUartRx\_byteAvailable
  - vpUart.h, 97
- vpUartRx\_hsOff
  - vpUart.h, 97
- vpUartRx\_hsOn
  - vpUart.h, 97
- vpUartRx\_ISR
  - vpUart.h, 95
- vpUartRx\_parityError
  - vpUart.h, 97
- vpUartRx\_receiveByte
  - vpUart.h, 98
- vpUartTx
  - vpUart.h, 95
- vpUartTx\_ISR
  - vpUart.h, 96
- vpUartTx\_ready
  - vpUart.h, 98
- vpUartTx\_sendByte
  - vpUart.h, 96
- vpUninstall
  - vplib.h, 100
- writePinDirection
  - portpin.h, 55
- writePinLatch
  - portpin.h, 56
- writePortDirection
  - portpin.h, 56
- writePortLatch
  - portpin.h, 56
- x00
  - memory.h, 47
- x0C
  - memory.h, 47
- x0D
  - memory.h, 47
- x0E
  - memory.h, 47
- xF5
  - memory.h, 47
- xF6
  - memory.h, 47
- xF7
  - memory.h, 47
- xF8
  - memory.h, 48
- xF9
  - memory.h, 48
- atoi
  - stdlib.h, 79