NAME

microseq - ppbus microsequencer developer's guide

SYNOPSIS

#include <sys/types.h>
#include <dev/ppbus/ppbconf.h>
#include <dev/ppbus/ppb_msq.h>

DESCRIPTION

See ppbus(4) for ppbus description and general info about the microsequencer.

The purpose of this document is to encourage developers to use the microsequencer mechanism in order to have:

1. a uniform programming model

2. efficient code

Before using microsequences, you are encouraged to look at ppc(4) microsequencer implementation and an example of how using it in ppi(4).

PPBUS register model

Background

The parallel port model chosen for ppbus is the PC parallel port model. Thus, any register described later has the same semantic than its counterpart in a PC parallel port. For more info about ISA/ECP programming, get the Microsoft standard referenced as "Extended Capabilities Port Protocol and ISA interface Standard". Registers described later are standard parallel port registers.

Mask macros are defined in the standard ppbus include files for each valid bit of parallel port registers.

Data register

In compatible or nibble mode, writing to this register will drive data to the parallel port data lines. In any other mode, drivers may be tri-stated by setting the direction bit (PCD) in the control register. Reads to this register return the value on the data lines.

Device status register

This read-only register reflects the inputs on the parallel port interface.

Bit	*Name*	*Description*
7	nBUSY	

inverted version of parallel port Busy signal

6 nACK version of parallel port nAck signal

5 PERROR
version of parallel port PERROR signal

4 SELECT
version of parallel port Select signal

3 nFAULT
version of parallel port nFault signal

Others are reserved and return undefined result when read.

Device control register

This register directly controls several output signals as well as enabling some functions.

Bit	*Name*	*Description*
5	PCD	direction bit in extended modes
4	IRQENABLE	1 enables an interrupt on the rising edge of nAck
3	SELECTIN	inverted and driven as parallel port nSelectin signal
2	nINIT	driven as parallel port nInit signal
1	AUTOFEED	inverted and driven as parallel port nAutoFd signal
0	STROBE	inverted and driven as parallel port nStrobe signal

MICROINSTRUCTIONS

Description

Microinstructions are either parallel port accesses, program iterations, submicrosequence or C calls. The parallel port must be considered as the logical model described in ppbus(4).

Available microinstructions are:

```
#define MS_OP_GET      0   /* get <ptr>, <len>                          */
#define MS_OP_PUT      1   /* put <ptr>, <len>                          */
#define MS_OP_RFETCH   2          /* rfetch <reg>, <mask>, <ptr>                  */
#define MS_OP_RSET     3          /* rset <reg>, <mask>, <mask>                   */
#define MS_OP_RASSERT  4          /* rassert <reg>, <mask>              */
#define MS_OP_DELAY    5          /* delay <val>                                  */
#define MS_OP_SET      6   /* set <val>                                 */
#define MS_OP_DBRA     7 /* dbra <offset>                             */
```

```
#define MS_OP_BRSET     8 /* brset <mask>, <offset>                  */
#define MS_OP_BRCLEAR   9       /* brclear <mask>, <offset>                */
#define MS_OP_RET       10 /* ret <retcode>                           */
#define MS_OP_C_CALL    11      /* c_call <function>, <parameter>      */
#define MS_OP_PTR       12      /* ptr <pointer>                       */
#define MS_OP_ADELAY    13      /* adelay <val>                                */
#define MS_OP_BRSTAT    14      /* brstat <mask>, <mask>, <offset>     */
#define MS_OP_SUBRET    15      /* subret <code>                       */
#define MS_OP_CALL      16      /* call <microsequence>                        */
#define MS_OP_RASSERT_P         17      /* rassert_p <iter>, <reg>             */
#define MS_OP_RFETCH_P  18      /* rfetch_p <iter>, <reg>, <mask>      */
#define MS_OP_TRIG      19      /* trigger <reg>, <len>, <array>       */
```

Execution context

The *execution context* of microinstructions is:

- the *program counter* which points to the next microinstruction to execute either in the main microsequence or in a subcall
- the current value of *ptr* which points to the next char to send/receive
- the current value of the internal *branch register*

This data is modified by some of the microinstructions, not all.

MS_OP_GET and MS_OP_PUT

are microinstructions used to do either predefined standard IEEE1284-1994 transfers or programmed non-standard io.

MS_OP_RFETCH - Register FETCH

is used to retrieve the current value of a parallel port register, apply a mask and save it in a buffer.

Parameters:

1. register
2. character mask
3. pointer to the buffer

Predefined macro: MS_RFETCH(reg,mask,ptr)

MS_OP_RSET - Register SET
is used to assert/clear some bits of a particular parallel port register, two masks are applied.

Parameters:

1. register
2. mask of bits to assert
3. mask of bits to clear

Predefined macro: MS_RSET(reg,assert,clear)

MS_OP_RASSERT - Register ASSERT
is used to assert all bits of a particular parallel port register.

Parameters:

1. register
2. byte to assert

Predefined macro: MS_RASSERT(reg,byte)

MS_OP_DELAY - microsecond DELAY
is used to delay the execution of the microsequence.

Parameter:

1. delay in microseconds

Predefined macro: MS_DELAY(delay)

MS_OP_SET - SET internal branch register
is used to set the value of the internal branch register.

Parameter:

1. integer value

Predefined macro: MS_SET(accum)

MS_OP_DBRA - Do BRAnch
is used to branch if internal branch register decremented by one result value is positive.

Parameter:

1. integer offset in the current executed (sub)microsequence. Offset is added to the index of the next microinstruction to execute.

Predefined macro: MS_DBRA(offset)

MS_OP_BRSET - BRanch on SET
is used to branch if some of the status register bits of the parallel port are set.

Parameter:

1. bits of the status register

2. integer offset in the current executed (sub)microsequence. Offset is added to the index of the next microinstruction to execute.

Predefined macro: MS_BRSET(mask,offset)

MS_OP_BRCLEAR - BRanch on CLEAR
is used to branch if some of the status register bits of the parallel port are cleared.

Parameter:

1. bits of the status register

2. integer offset in the current executed (sub)microsequence. Offset is added to the index of the next microinstruction to execute.

Predefined macro: MS_BRCLEAR(mask,offset)

MS_OP_RET - RETurn
is used to return from a microsequence. This instruction is mandatory. This is the only way for the

microsequencer to detect the end of the microsequence. The return code is returned in the integer pointed by the (int *) parameter of the ppb_MS_microseq().

Parameter:

1. integer return code

Predefined macro: MS_RET(code)

MS_OP_C_CALL - C function CALL

is used to call C functions from microsequence execution. This may be useful when a non-standard i/o is performed to retrieve a data character from the parallel port.

Parameter:

1. the C function to call

2. the parameter to pass to the function call

The C function shall be declared as a *int(*)(void *p, char *ptr)*. The ptr parameter is the current position in the buffer currently scanned.

Predefined macro: MS_C_CALL(func,param)

MS_OP_PTR - initialize internal PTR

is used to initialize the internal pointer to the currently scanned buffer. This pointer is passed to any C call (see above).

Parameter:

1. pointer to the buffer that shall be accessed by xxx_P() microsequence calls. Note that this pointer is automatically incremented during xxx_P() calls

Predefined macro: MS_PTR(ptr)

MS_OP_ADELAY - do an Asynchronous DELAY

is used to make a tsleep() during microsequence execution. The tsleep is executed at PPBPRI level.

Parameter:

1. delay in ms

Predefined macro: MS_ADELAY(delay)

MS_OP_BRSTAT - BRanch on STATe
is used to branch on status register state condition.

Parameter:

1. mask of asserted bits. Bits that shall be asserted in the status register are set in the mask
2. mask of cleared bits. Bits that shall be cleared in the status register are set in the mask
3. integer offset in the current executed (sub)microsequence. Offset is added to the index of the next microinstruction to execute.

Predefined macro: MS_BRSTAT(asserted_bits,clear_bits,offset)

MS_OP_SUBRET - SUBmicrosequence RETurn
is used to return from the submicrosequence call. This action is mandatory before a RET call. Some microinstructions (PUT, GET) may not be callable within a submicrosequence.

No parameter.

Predefined macro: MS_SUBRET()

MS_OP_CALL - submicrosequence CALL
is used to call a submicrosequence. A submicrosequence is a microsequence with a SUBRET call.
Parameter:

1. the submicrosequence to execute

Predefined macro: MS_CALL(microseq)

MS_OP_RASSERT_P - Register ASSERT from internal PTR
is used to assert a register with data currently pointed by the internal PTR pointer. Parameter:

1. amount of data to write to the register
2. register

Predefined macro: MS_RASSERT_P(iter,reg)

MS_OP_RFETCH_P - Register FETCH to internal PTR

is used to fetch data from a register. Data is stored in the buffer currently pointed by the internal PTR pointer. Parameter:

1. amount of data to read from the register
2. register
3. mask applied to fetched data

Predefined macro: MS_RFETCH_P(iter,reg,mask)

MS_OP_TRIG - TRIG register

is used to trigger the parallel port. This microinstruction is intended to provide a very efficient control of the parallel port. Triggering a register is writing data, wait a while, write data, wait a while... This allows to write magic sequences to the port. Parameter:

1. amount of data to read from the register
2. register
3. size of the array
4. array of unsigned chars. Each couple of u_chars define the data to write to the register and the delay in us to wait. The delay is limited to 255 us to simplify and reduce the size of the array.

Predefined macro: MS_TRIG(reg,len,array)

MICROSEQUENCES

C structures

```
union ppb_insarg {
    int     i;
    char    c;
    void    *p;
    int     (* f)(void *, char *);
};
```

```
struct ppb_microseq {
    int                 opcode;         /* microins. opcode */
    union ppb_insarg    arg[PPB_MS_MAXARGS];    /* arguments */
};
```

Using microsequences

To instantiate a microsequence, just declare an array of ppb_microseq structures and initialize it as needed. You may either use predefined macros or code directly your microinstructions according to the ppb_microseq definition. For example,

```
    struct ppb_microseq select_microseq[] = {

                /* parameter list
                 */
                #define SELECT_TARGET    MS_PARAM(0, 1, MS_TYP_INT)
                #define SELECT_INITIATOR MS_PARAM(3, 1, MS_TYP_INT)

                /* send the select command to the drive */
                MS_DASS(MS_UNKNOWN),
                MS_CASS(H_nAUTO | H_nSELIN |  H_INIT | H_STROBE),
                MS_CASS( H_AUTO | H_nSELIN |  H_INIT | H_STROBE),
                MS_DASS(MS_UNKNOWN),
                MS_CASS( H_AUTO | H_nSELIN | H_nINIT | H_STROBE),

                /* now, wait until the drive is ready */
                MS_SET(VP0_SELTMO),
/* loop: */     MS_BRSET(H_ACK, 2 /* ready */),
                MS_DBRA(-2 /* loop */),
/* error: */    MS_RET(1),
/* ready: */    MS_RET(0)
    };
```

Here, some parameters are undefined and must be filled before executing the microsequence. In order to initialize each microsequence, one should use the ppb_MS_init_msq() function like this:

```
    ppb_MS_init_msq(select_microseq, 2,
                            SELECT_TARGET, 1 << target,
                            SELECT_INITIATOR, 1 << initiator);
```

and then execute the microsequence.

The microsequencer

The microsequencer is executed either at ppbus or adapter level (see ppbus(4) for info about ppbus system layers). Most of the microsequencer is executed at ppc level to avoid ppbus to adapter function call overhead. But some actions like deciding whereas the transfer is IEEE1284-1994 compliant are executed at ppbus layer.

SEE ALSO

ppbus(4), ppc(4), ppi(4)

HISTORY

The **microseq** manual page first appeared in FreeBSD 3.0.

AUTHORS

This manual page was written by Nicolas Souchu.

BUGS

Only one level of submicrosequences is allowed.

When triggering the port, maximum delay allowed is 255 us.