WiCAN Library

Manual

Version 1.0.4
2001-05-14

Copyright

LAWICEL HB
Klubbgatan 3
SE-282 32 TYRINGE
SWEDEN

Phone: +46 (0)451 59877
Fax: +46 (0)451 59878

http://www.lawicel.com
info@lawicel.com

http://www.candip.com
info@candip.com

1 Introduction

The WiCAN library supports the stand-alone CAN controller SJA1000 from Philips together with an Atmel AVR (type AT90S8515 or Atmega that can work with external XRAM).

The WiCAN library offers functions for the configuration of the CAN controller as well as functions for transmission and reception of CAN objects through FIFO queues. The library can handle the standard 11 bit identifiers as well as the extended 29bit identifiers.

The following AVR's are supported through this version of the WiCAN library:

- AT90S8515 (8k FLASH, 515b RAM & 512b EEPROM).
- And is specially design to work with the CANDIP/AVR1 from LAWICEL.

The following Compilers are supported through this vesion of the WiCAN library:

- ImageCraft ICCAVR Version 6.21 or later.

2 Installion procedure

The installation is rather easy. Simply copy the files on the disk on to the hard disk and read The readme.txt file for information on how to install the WiCAN LIB and make your first test application.

3 Licence

By installing the WiCAN library, the user agrees only to use this software for their own purpose and not sell it further on as source code.

4 Remarks

In the header file WiCAN.H are all definitions which are important for an application. It is only that header file that has to be included in the application together with the WiCAN.C file if the LIB is bought as source code, otherwise the LIB has to be included in the project.

5 Structure of the library

The data structure of the WiCAN library is described below.

The basic idea of the library is to provide an Application interface (API) to the CAN controller. If the CAN controller is a BasicCAN controller, there are 2 message queues, one for incomming CAN messages and one for outgoing messages. The structure for each frame is built up like this and takes only 14bytes of RAM per frame.

```
typedef struct {
    _U32 id;
    _U08 len;
    _U08 flags;
    _U08 byte[8];
  } WiCAN_Object;
```

In the WiCAN.H header file there are also some standard definitions for variables:

```
#define  _U08     unsigned char
#define  _S08     char
#define  _U16     unsigned int
#define  _S16     int
#define  _U32     unsigned long
#define  _S32     long
```

6 Library functions

6.1 Initialization

6.1.1 WiCAN_Init

Usage:

Used for setting the configuration/initialization of the CAN controller. When this function has been called, the controller is set into the reset/init state.

Calling:

```
byte_result = WiCAN_Init(_U08 StdSpeed, _U08 Btr0, _U08 Btr1,
                         _U08 Ocr);
```

Parameters:

StdSpeed	Standard speed according to CiA draft standard 102.
	Use WiCAN_SPEED_MANUAL to set
	Btr0 and Btr1 manually.
	
	`WiCAN_SPEED_10K`
	`WiCAN_SPEED_20K`
	`WiCAN_SPEED_50K`
	`WiCAN_SPEED_100K`
	`WiCAN_SPEED_125K`
	`WiCAN_SPEED_250K`
	`WiCAN_SPEED_500K`
	`WiCAN_SPEED_800K`
	`WiCAN_SPEED_1M`
	`WiCAN_SPEED_MANUAL`

Btr0, Btr1	Registers for setting of Synchronization jump width and Baud Rate Prescaler. Set these parameters to zero when a standard speed is selected.
Ocr	Output Control register for the CAN controller.

Return values:

0	CAN Controller is in reset/init state. `WiCAN_OK`
1	Parameters are wrong or not supported. `WiCAN_INIT_SPEED_ERR`

6.1.2 WiCAN_Start

Usage:

Used for starting the CAN controller (i.e. reception/transmisson is possible). When this function has been called, the controller is set into the run state.

Calling:

```
byte_result = WiCAN_Start(void);
```

Parameters:

None

Return value:

0	CAN Controller is in run state. `WiCAN_OK`
1	CAN Controller is not started, due to that it isn't initiated. `WiCAN_NOT_INITIATED`

6.1.3 WiCAN_Stop

Usage:

Used for stop the CAN controller (i.e. reception/transmisson is disabled). When this function has been called, the controller is set into the reset/init state.

Calling:

```
byte_result = WiCAN_Stop(void);
```

Parameters:

None

Return value:

0	CAN Controller is in reset/init state. `WiCAN_OK`
1	CAN Controller is not in run mode and can therefore not be stopped. `WiCAN_NOT_STARTED`

6.2 Transmit and Receive

6.2.1 WiCAN_EmptyOutQueue

Usage:

Clear the transmit buffer.

Calling:

```
byte_result = WiCAN_EmptyOutQueue();
```

Parameters:

none

Return value:

0	Message object enabled. `WiCAN_OK`
1	Error. `WiCAN_ERR`

6.2.2 WiCAN_EmptyInQueue

Usage:

Clear the receive buffer.

Calling:

```
byte_result = WiCAN_EmptyInQueue();
```

Parameters:

none

Return value:

0	Message object enabled. `WiCAN_OK`
1	Error. `WiCAN_ERR`

6.2.3 WiCAN_GetRxQueueSize()

Usage:

Retreives hom many CAN frames that are pending/waiting in the receive buffer.

Calling:

```
byte_result = WiCAN_GetRxQueueSize();
```

Parameters:

none

Return value:

Number of messages waiting.

6.2.4 WiCAN_GetTxQueueSize()

Usage:

Retreives hom many CAN frames that are pending/waiting in the transmit buffer.

Calling:

```
byte_result = WiCAN_GetTxQueueSize();
```

Parameters:

none

Return value:

Number of messages waiting.

6.2.5 WiCAN_SendFrame

Usage:

Puts a CAN frame into the transmit FIFO buffer. The message will be sent as soon as the CAN controller is ready and the bus is free.

Calling:

```
byte_result = WiCAN_SendFrame(WiCAN_Object_t *messobj);
```

Parameters:

messobj	pointer to the data structure of the CAN frame.

Return value:

0	New message has successfully been put into the buffer. `WiCAN_OK`
1	The FIFO queue is full. `WiCAN_TXBUF_FULL`

6.2.6 WiCAN_GetFrame

Usage:

Reads a CAN frame from the receive FIFO buffer (if there is one waiting).

Calling:

```
byte_result = WiCAN_GetFrame(WiCAN_Object_t *messobj);
```

Parameters:

messobj	pointer to the data structure.

Return value:

0	New message is availible. `WiCAN_OK`
1	No new message in queue. `WiCAN_RXBUF_EMPTY;`

6.3 Status etc.

6.3.1 WiCAN_GetStatus

Usage:

Get current status. (NOT IMPLEMENETED YET!)

Calling:

```
byte_result = WiCAN_GetStatus();
```

Parameters:

none

Return value:

6.3.2 WiCAN_GetVersion

Usage:

Get current version.

Calling:

```
byte_result = WiCAN_GetVersion();
```

Parameters:

none

Return value:

byte	Version number, see WiCAN.h for more info on valid versions pre-defined values. E.g.: `WiCAN_VER104`

6.4 Message Filtering (Acceptance Mask/Code)

6.4.1 WiCAN_SetAccCode

Usage:

Set Acceptance Code.

Calling:

```
byte_result = WiCAN_SetAccCode(_U08 Ac0,_U08 Ac1,_U08 Ac2,_U08 Ac3);
```

Parameters:

Ac0	AC0 value.
Ac1	AC1 value.
Ac2	AC2 value.
Ac3	AC3 value.

For more info on these values, see the Philips SJA1000 datasheet Section 6.4.15.2 “Dual filter configuration”. The WiCAN LIB uses This mode and it cannot be changed in the object version of the LIB.

Return value:

0	CAN Controller is in init state and call was OK. `WiCAN_OK`
1	CAN Controller is in run mode or not initiated. `WiCAN_ERR`

6.4.2 WiCAN_SetAccMask

Usage:

Set Acceptance Mask.

Calling:

```
byte_result = WiCAN_SetAccMask(_U08 Am0,_U08 Am1,_U08 Am2,_U08 Am3);
```

Parameters:

Am0	AM0 value.
Am1	AM1 value.
Am2	AM2 value.
Am3	AM3 value.

For more info on these values, see the Philips SJA1000 datasheet Section 6.4.15.2 “Dual filter configuration”. The WiCAN LIB uses This mode and it cannot be changed in the object version of the LIB.

Return value:

0	CAN Controller is in init state and call was OK. `WiCAN_OK`
1	CAN Controller is in run mode or not initiated. `WiCAN_ERR`