

Document: CLS626_SYS_V10X
Author : M.Majoor
Subject : Description CLS626.SYS driver Version 1.0x

Revision overview

Revision	Date	Author	Description
20020930	2002-09-30	MM	Initial release

Brief explanation of the CLS626 driver

Communication with the CLS626.SYS driver takes place using DeviceIO calls. This document describes these calls. Only the function and parameters passed/returned are discussed. Note that this driver is actually a SAA7146A driver and has no specifics towards the CLS626 board for which it is used for.

The driver will de-allocate all allocated resources only when the driver is shut down. This means that allocated memory keeps being allocated if not properly de-allocated.

The driver itself is relatively easy and only provides some basic support for handling interrupts. The main use of the driver is:

- . Reading/writing to SAA7146 registers
- . (de)allocation of contiguous 'DMA' memory
- . Reading/writing to this allocated 'DMA' memory. Note: the virtual address of the allocated memory can be used directly using the Windows 98 family. However, with W2000 this is not possible. For this there are two specific functions (one for reading, one for writing) which make reading/writing to this memory possible (V1.01 only; V1.00 does not have these two functions).
- . Triggering an event when an interrupt of the SAA7146 is activated

Version 2.00

There is also a version 2.00 of the driver. Version 2.00 is a commercial used version of the driver, which is used by Lismar Engineering B.V. For info on version 2.00 (obtaining the driver or code) of the driver you can email to m.majoor@lismar.com.

Version 2.00 has the following extensions:

- . More different DMA buffers possible (256 instead of 32)
- . Each of the 32 SAA7146 interrupts can be set to:
 - . Trigger the event (as V1.0x does but only selected interrupts will trigger it)
 - . Write specific data to a specified SAA7146 register (register is read and written back after an AND, OR, XOR operation is done on the data). This makes it possible to have, for instance, the SIGx signals being set/reset by an interrupt. A RPS program running in the SAA7146 can then use this as a trigger. This way each of the interrupts can be used in a RPS program.
 - . Buffering (copying) of allocated DMA memory on receipt of an interrupt. Multiple buffers for a single interrupt can be set. The driver advances to the next buffer after writing to the current buffer. Overflows and such are checked and reported.

IOCTL_GET_VERSION		
Get the version of the driver.		
IN	-	
OUT	USHORT majorversion	Major version
	USHORT minorversion	Minor version
	ULONG build	Build date \$YYYYMMDD YYYY = Year MM = Month DD = Day

IOCTL_GET_STATUS		
Get status information.		
IN	ULONG DMAbuffer	DMA buffer to return status for.
OUT	ULONG interrupts	Interrupts detected (total interrupts for driver).
	ULONG isr	Interrupt status register of last occurred interrupt. (SAA7146A register \$10C)

	PVOID vaBuffer	Virtual address DMA buffer. For Windows 98 you can use this address to directly access the allocated memory.
	PHYSICAL_ADDRESS paBuffer	Physical address DMA buffer. You need this address when a RPS program is used (a RPS program is a small program which can run inside the SAA7146A).
	ULONG buffersize	Size of DMA buffer in bytes.

IOCTL_SAA7146_READ		
Read register from SAA7146.		
IN	ULONG address	Address of the register from the SAA7146 to read.
OUT	ULONG data	Data read from register.

IOCTL_SAA7146_WRITE		
Write data to a register of the SAA7146.		
IN	ULONG address	Address of the register from the SAA7146 to write to.
	ULONG data	Data to write to the register.
OUT	-	

IOCTL_WAIT_NOTIFY		
Wait for a notification (interrupt). Only when an interrupt is generated this call will return. Thus, this call waits until this occurs. The only way to recover for this, besides an actual interrupt, is to manually trigger an event using IOCTL_GENERATE_EVENT. Note: the driver only supports a single IOCTL_WAIT_NOTIFY. If more than a single IOCTL_WAIT_NOTIFY is issued only the first is 'acknowledged'. The others will wait for the next generated event.		
IN	-	
OUT	ULONG bogus	Nothing is actually being returned.

IOCTL_GENERATE_EVENT		
Generate a manual event. Used to end an IOCTL_WAIT_NOTIFY manually.		
IN	ULONG bogus	Not used.
OUT	-	

IOCTL_DMA_ALLOCATE		
Allocate memory which can be used for DMA purposes. The memory allocated is a single contiguous block of memory. The driver allows a maximum of 32 buffers to be allocated.		
IN	ULONG buffersize	Size in bytes of contiguous memory to allocate.
OUT	LONG bufferId	Identification of buffer. This identification is used to de-allocate the memory or getting status information for it.

	PVOID vaBuffer	Virtual address of buffer. Note: Window 98 allows using this address to access the allocated memory directly. However, this does not work with W2000 (use IOCTL_DMA_READ). It is advised, for Windows 98 also, to use IOCTL_DMA_READ and IOCTL_DMA_WRITE to access the allocated memory.
	PHYSICAL_ADDRESS paBuffer	Physical address of buffer. You need this address when a RPS program is used (a RPS program is a small program which can run inside the SAA7146A).
	ULONG buffersize	Size in bytes of buffer

IOCTL_DMA_RELEASE		
Release the memory previously allocated.		
IN	LONG bufferId	Identification of buffer. This is the identification returned when IOCTL_DMA_ALLOCATE is used.
OUT	-	

IOCTL_DMA_READ (not available in V1.00)		
Read from IOCTL_DMA_ALLOCATE allocated memory.		
IN	LONG bufferId	Identification of buffer. This is the identification returned when IOCTL_DMA_ALLOCATE is used.
	PUCHAR bufferTransfer	Pointer to target buffer
	ULONG bufferSourceIndex	Index in DWORDS (4 bytes) into source (DMA memory) to start copying from.
	ULONG bufferTargetIndex	Index in DWORDS into target buffer to start copying to.
	ULONG bufferTransferLength	Number of DWORDS to transfer.
OUT	-	

IOCTL_DMA_WRITE (not available in V1.00)		
Write to IOCTL_DMA_ALLOCATE allocated memory.		
IN	LONG bufferId	Identification of buffer. This is the identification returned when IOCTL_DMA_ALLOCATE is used.
	PUCHAR bufferTransfer	Pointer to source buffer
	ULONG bufferSourceIndex	Index in DWORDS (4 bytes) into source buffer to start copying from.
	ULONG bufferTargetIndex	Index in DWORDS into target buffer (DMA memory) to start copying to.
	ULONG bufferTransferLength	Number of DWORDS to transfer.
OUT	-	