APPLICATION NOTE SOFTWARE

UAC 357xB
Programmer's Guide

Edition ...
6251-5XX-XAS

MICRONAS

Contents

Contents, continued

Programmer's Guide

1. Introduction

The basic information on functionality and electrical characteristics is given in the data sheet UAC 357xB. This is based on the Micronas standard firmware or SDK firmware, which is a good example for a feature set needed in a general purpose USB audio codec.

The flexibility of the device, however, allows customization to match a specific application exactly. This is achieved by the use of a programmable internal microcontroller and supply of a software development kit (SDK).

The purpose of this paper is to provide all information which is needed to customize the UAC357xy firmware.

Beginning with general information on the SDK environment and usage, an explanation of the standard firmware structure is given. This is followed by the detailed description of all functions which may be modified on customization, like default values, descriptors and all audio class related items. This tutorial is completed with register lists for both V8-microcontroller and DSP.

Information on the USB chapter 9 handling is not given because this routines are strictly tied to the hardware of the chip and it is not recommended to modify this blocks. It is also not possible to modify the DSP firmware.

2. V8 Firmware Source Files

The Micronas-Standard-Firmware package consists of the following files:

Table 2–1: Source Files

File Name	Description
data_fixed_ram.asm	RAM variables with fixed address
data_sec1.asm	Section1 variables
dsp_control.asm	controls the audio functions in the DSP
emu_flash.asm	structure for the I2C-EEPROM
gpio_control.asm	GPIO communicaton and the HID
i2c_drv.asm	I2C driver
iso_down_as1.asm	alternate setting 1 for downstream
iso_down_as2.asm	alternate setting 2 for downstream
iso_down_as3.asm	alternate setting 3 for downstream
iso_up_as1.asm	alternate setting 1 for upstream
iso_up_as2.asm	alternate setting 2 for upstream
iso_up_as3.asm	alternate setting 3 for upstream
macros.asm	v8 macros
main.asm	main program – includes all other .asm`s
main_config.asm	memory map of the V8 address space
main_include.asm	wakeup routines, initialisations
main_upper_rom.asm	content of upper rom space
timer_drv.asm	Timer driver
usb_appl_interface.asm	application specific chapter 9 stuff
usb_audio_ctl.asm	handles audio class requests

Table 2–1: Source Files

File Name	Description
usb_audio_ctl_include.asm	routines and tables for audio class requests
usb_ch9.asm	application independant chapter 9 handling
usb_class.asm	branch handler to class spec. requests
usb_descriptor_xp.asm	Standard Descriptor („WINXP-proved“)
usb_hid_ctl.asm	empty – handles hid class requests
usb_vend_class.asm	empty – handles vendor class requests
usb_vendor.asm	handles vendor spec chapter 9 requests

The following diagram displays how all these files are related to each other:

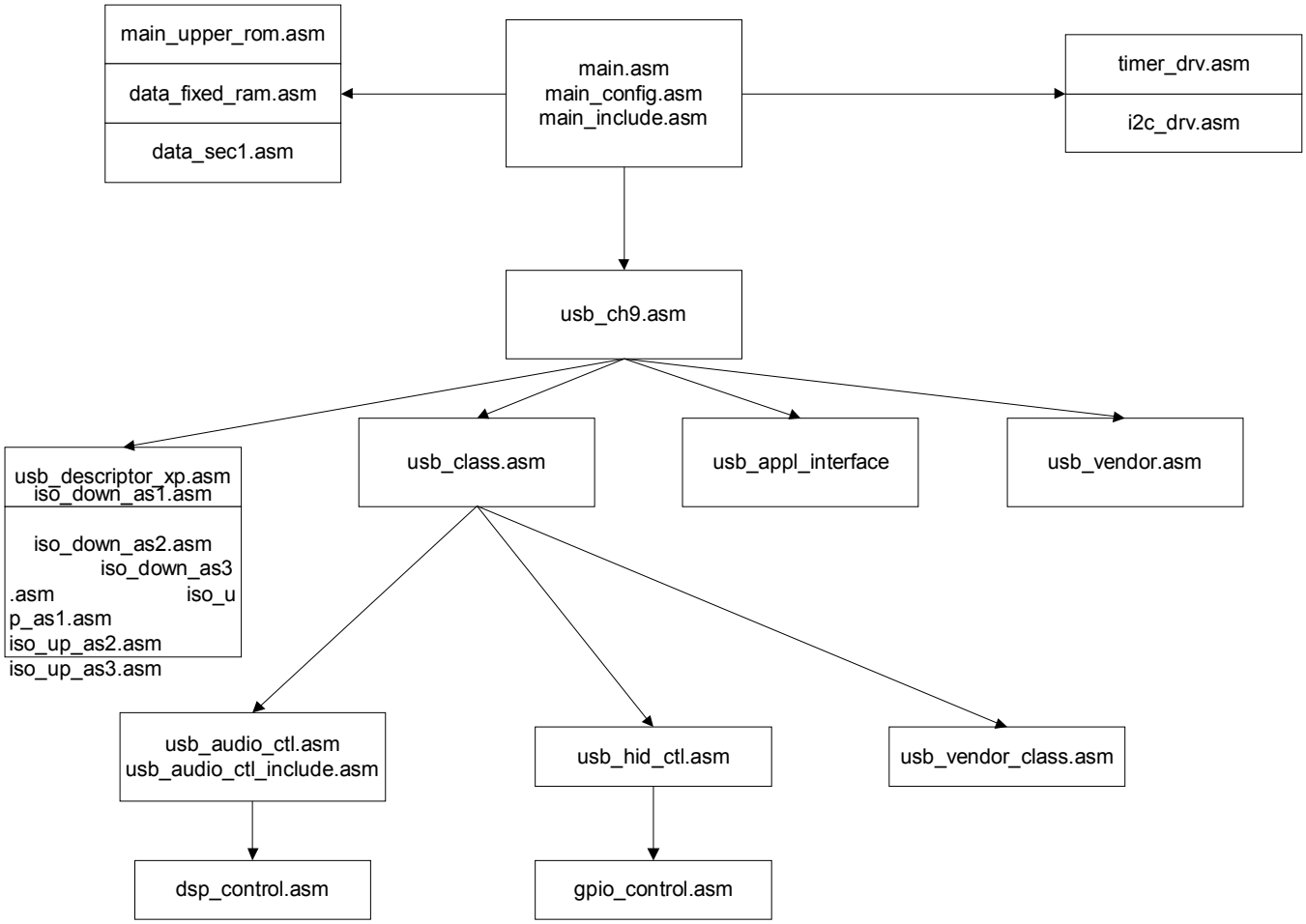

Fig. 2–1: Firmware overview

3. Bootloader Concept

-> von W. Platzer bis KW 25

3.1. External EEPROM

3.1.1. Introduction

The UAC 357xB can load its configuration and plug-ins out of an external I^2C EEPROM.

3.1.2. EEPROM Types

The UAC 357xB can have different EEPROMS connected to the I^2C bus. The UAC 357xB is working as an I^2C bus master at this point in time. Depending on EEPROM size the EEPROM can hold different content (see Table 3–1)

Table 3–1: Supported I^2C EEPROM types

EEPROM size	Purpose
2kbit	Configuration only
4-32kbit	Configuration Plug-in software
64kbit	Configuration On reset loadable firmware
128kbit	Configuration On reset loadable firmware Plug-in software

3.1.3. EEPROM bootstrap options

The EEPROM configuration can be set with bootstrap options at the pins USBDAT, USBCLK and USBWSO (see Table 3–2).

Table 3–2:

USBWSO	USBDAT	USBCLK	Address Subaddress	Purpose
1	1	don't care		internal ROM only I^2C master disabled
1	0	don't care	0x50 1byte subaddressing	Configuration data Plug-in Software 100kHz I^2C master
0	1	0	0x51 2byte subaddressing	Configuration data On reset loadable firmware Plug-in software 400kHz I^2C
0	1	1	0x52 2byte subaddressing	Configuration data On reset loadable firmware Plug-in software 400kHz I^2C
0	0	0	0x51 2byte subaddressing	Configuration data On reset loadable firmware Plug-in software 100kHz I^2C
0	0	1	0x52 2byte subaddressing	Configuration data On reset loadable firmware Plug-in software 100kHz I^2C

3.1.4. EEPROM content

3.1.4.1. Header

The first byte in the serial EEPROM describes the EEPROM content (see Table 3–3).

Table 3–3: EEPROM configuration byte

bit	function
bit[7]	verify checksum
bit[3..6]	must be set to 0x00
bit[2]	new firmware
bit[1]	plug-in
bit[0]	UAC357xB configuration

This allows to have 3 different sections in the EEPROM. The address and length of all this sections are stored in the following locations in the EEPROM.

3.1.5. EEPROM Configuration Section

Table 3–4: EEPROM configuration section

Address	Function
0x0000	EEPROM configuration
0x0001 / 0x0002	Configuration section length High Byte / Low Byte
0x0003 / 0x0004	Configuration section address High Byte / Low Byte
0x0005 / 0x0006	Plug-in section length High Byte / Low Byte
0x0007 / 0x0008	Plug-in section address High Byte / Low Byte
0x0009 / 0x000A	Firmware section length High Byte / Low Byte
0x000B / 0x000C	Firmware section address High Byte / Low Byte

3.1.5.1. I/O and HID

Address	Function	Name
0x00	MIN_PB_VOLUME bit [7..0] volume of the DAC output after configuration: 0x00 0dB 0xEC -20dB gets overwritten by Set_Volume requests later on	MIN_PB_VOLUME
0x01	reserved 2 bit [7..0] reserved must be set to 0x00	Reserved_2
0x02	**V8_CTL_DEFAULT** bit [7]set to 0 for GPIO[11] standard output set to 1 for GPIO[11] enable PWM output bit [6..0] reserved must be set to 0x08	V8_CTL_DEFAULT
0x03	**reserved 3** bit [7..0] reserved must be set to 0x01	Reserved_3
0x04	**reserved 4** bit [7..0] reserved must be set to 0x09	Reserved_4

Address	Function	Name
0x05	**reserved 5** bit [7..0] reserved must be set to 0x05	Reserved_5
0x06	**reserved 6** bit [7..0] reserved must be set to 0x0b	Reserved_6
0x07	**reserved 7** bit [7..0] reserved must be set to 0x04	Reserved_7
0x08	**Miscellaneous configuration 1** bit [7] 0: µC uses USBWSO, USBDAT, USBCLK interface 1: DSP I²S out uses USBWSO, USBDAT, USBCLK interface bit [6..2] reserved must be set to MSB 01000 LSB bit [1..0] selects audio oversampling clock frequency pin MCLK 00: 36.8MHz 01: 24.576MHz 10: 18.432MHz 11: do not use	Config_Misc_1
0x09	**Analog Control** bit[7] outlron : 0 : disable headphone opamp 1 : enable headphone opamp, force sref to on state (default) bit[6] srefon : 0 : sref off 1 : sref on (default) bit[5] filton : 0 : disable filter opamp 1 : enable filter opamp bit[4] reserved, must be set to zero bit[3] Internal reset enable bit[2] Pseudo differential output mode bit[1] Common output mode bit[0] setagn : set voltage of sref 0 : 1.725V 1 : 2.3V	ANCTR
0x0a	**GPIO[7..0] direction** 0: input 1: output bit [7..0] set GPIO [7..0] to input/output	PIO
0x0b	**GPIO[11..8] direction** bit [7..4] reserved must be set to MSB 0000 LSB 0: input 1: output bit [3..0] set GPIO [11..8] to input/output	PIO2

Address	Function	Name
0x0c	**GPIO[7..0] pulldown enable** 0: disable pull-down 1: enable pull-down bit [7..0] GPIO [7..0] pull-down resistor enable / disable	PDEN
0x0d	**GPIO[11..8] pull-down enable** bit [7..4] reserved must be set to MSB 0000 LSB 0: disable pull-down 1: enable pull-down bit [3..0] GPIO [11..8] pull-down resistor enable / disable	PDEN2
0x0e	**GPIO[7..0] driver strength** 0: weak 1: strong bit [7..0] GPIO [7..0] driver strength weak / strong	PS
0x0f	**GPIO[11..8] driver strength** 0: weak 1: strong bit [7] Pins: DAO, DAI, WSI, CLI driver strength weak / strong bit [6] Pins: USBWSO, USBCLK, USBDAT driver strength weak / strong bit [5] Pins: SOF, SEN, SUSPENDQ, TRDY driver strength weak / strong bit [4] Pins: STRB, RD DRIVER STRENGTH weak / strong bit [3..0] GPIO [11..8] driver strength weak / strong	PS2
0x10	**IO Config 1** 0 : input / tristate 1 : output bit [7] Direction of pin: STRB bit [6] Direction of pin: RD bit [5] Direction of pin: USBDAT bit [4] Direction of pin: USBCLK bit [3] reserved set to 0: bit [2] Direction of pin: CLI bit [1] Direction of pin: WSI bit [0] Direction of pin: DAI	IO_CONFIG_1

Address	Function	Name
0x11	**IO Config 2** 0 : input / tristate 1 : output bit [7] Direction of pin: MCLK bit [6] Direction of pin: DAO bit [5] Direction of pin: USBWS bit [4] Direction of pin: SEN 0: disable pull-down resistor 1: enable pull-down resistor bit [3] enable pull-down resistor of pin VBUS: 0 : input / tristate 1 : output bit [2] Direction of pin: TRDY bit [1] Direction of pin: SOF bit [0] Direction of pin: SUSPENDQ	IO_CONFIG_2
0x12	**reserved 8** bit [7:0] reserved set to 0xa7	reserved_8
0x13	**reserved 9** bit [7:0] reserved set to 0x91	reserved_9
0x14	**Key timer** Timer value for debouncing of the HID keys bit [7:0] set to a value between 0x01 and 0xff; default 0x7f	reserved_8
0x15 - 0x44	**equalizer coefficients** coefficients for the 5-band parametric equalizer	
0x45 - 0x47	**I²S configuration** for a detailed description see register description of the DSP	i2S_CONFIG
0x48 - 0x49	**I²S record selection** for a detailed description see register description of the DSP	i2S_CONFIG
0x4A - 0x50	**Bass Boost On Setting** **Micronas Bass settings for Bass Boost On** for a detailed description see Micronas Bass for UAC357xB Application note	BassBoostOn
0x51 - 0x57	**Bass Boost Off Setting** **Micronas Bass settings for Bass Boost On** for a detailed description see Micronas Bass for UAC357xB Application note	BassBoostOff
0x58	**AGC decay** sets the AGC decay time for the DAC path. For more information see the the register description of the DSP	AgcDecay
0x59	**DSP mode selection** selects the required mode for the DSP. Full feature mode(72MHz) or reduced feature mode(48Mhz). 0: Full feature mode(72MHz) 1: Reduced feature mode(48MHz	DspMode

Address	Function	Name
0x5a	**DAC on/off** Turns the required bit [7:2]reserved should be set to 0 bit [1] reserved bit [0] 0: Stereo DAC Off 1: Stereo DAC On	AgcDecay
0x5b	**reserved 10** reserved set to 0x00	reserved_10
0x5c - 0x5d	**Vendor ID** the USB vendor ID. e.g. Micronas has 0x074d byte order low-byte / high-byte	VendorId
0x5e - 0x5f	**Product ID** the USB product ID. e.g. UAC3576B has 0x3576 byte order low-byte / high-byte	ProductId
0x60 - 0x61	**Device release code** **BCD number!!**: use only binary coded decimals here. Using hex numbers will confuse hosts. order low-byte / high-byte	bcdDevice
0x62	**iManufacture** index number of the manufacture string in the EEPROM. valid numbers 0..3 0: no string in the table 1,2,3: string number in the table	iManufacture
0x63	**iProduct** index number of the product string in the EEPROM. valid numbers 0..3 0: no string in the table 1,2,3: string number in the table	iProduct
0x64	**iSerialNumber** index number of the serial number string in the EEPROM. valid numbers 0..3 0: no string in the table 1,2,3: string number in the table	iserial number
0x65	**iConfig** index number of the configuration string in the EEPROM. valid numbers 0..3 0: no string in the table 1,2,3: string number in the table	iConfig
0x66	**bmAttributes** reserved set to 0x00	bmAttributes
0x67	**Max power** reserved set to 0x00	MaxPower
0x68	**Interval** reserved set to 0x00	bInterval

Address	Function	Name
0x69 - 0x6a	**Pointer string 0** value: 0xa771	pString0
0x6b - 0x6c	**Pointer string 1** value: 0xa775	pString1
0x6d - 0x6e	**Pointer string 2** value: pString1 + *pString1	pString2
0x6f - 0x70	**Pointer string 3** value: pString1 + *pString1 + *pString2	pString3
0x71 - 0x74	**String 0** byte 0: 0x04 byte 1: 0x03 byte 1: 0x09 byte 1: 0x04	String0
0x75 - 0xXX	**Strings 1..3** all strings are currently ASCII. No Unicode support. String format: byte 0: length of ASCII string + 2 byte 1: 0x03 byte 2..n ASCII string	String1 String2 String3

3.1.5.2. The Section1 Concept – data_sec1.asm

The Section1 is the first section of an optionally attached external EEPROM and is used to hold all variables and default values that may be different from one application to the other, based on the same internal ROM-firmware. The standard firmware handles 256-bytes Section1 data.

All data are defined in data_sec1.asm. All data names here have the prefix „SED_DEFAULT_ „

(originally an sed-script was used to handle this file).

This file is included at the position 0x2E00 of the upper-ROM and is the default Section1 of a ROM firmware. The assembler script generates the file „data_sec1ram.asm“ which is an mirror image of the Section1 in the RAM area at 0xA700 by just inserting placeholders using the same names apart from the SED_DEAULT prefix. The scripts also generates the file „emu_section1.asm“ which is a hex-representation of the data_sec1.asm. This file is then included in the „emu_flash.asm“. This serves as the default Section1 code in the external EEPROM. So without changing ROM firmware the default setting and the descriptor IDs and strings can be modified.

Note: It is not allowed to change the length or the ordering of the Section1 data in the EEPROM. Length and ordering must be identical with that of the upper ROM Section1. Descriptor strings are not affected by the length-restriction as long as the 256-byte limit is not violated.

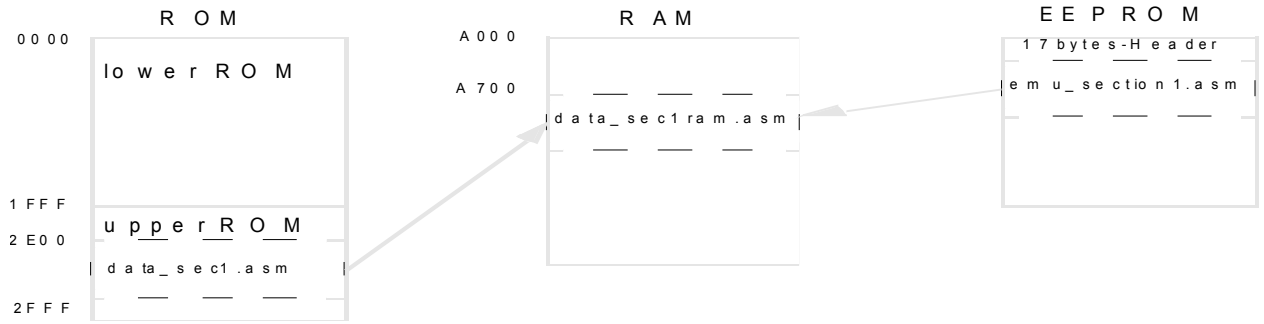

Fig. 3–1:

Note: The bootloader (see „The Bootloader Concept“) loads the data_sec1.asm from the upper ROM into the placeholder-RAM area and overwrites this area with Section1 - data coming from the external EEPROM (if an EEPROM with Section1 data is connected.

The Section1 concept is targeted for a device without shadow RAM. Applications based on devices with Shadow RAM can keep all Section1 data as part of the emulatable lower ROM, of course.

3.1.6. External EEPROM

The UAC 357xB is equipped with it's own ROM including the Micronas firmware necessary for operation.

Optionally, the IC can be operated with the followings oft ware configurations:

1. Micronas firmware (internal ROM)
2. customer-specific firmware (external EEPROM)
3. Micronas firmware plus customer-specific plug-in code
4. Structure of the external EEPROM

The bootloader handles an external I^2C-EEPROM with automatic detection of presence and structure and download of code and data from up to four sections.

The header byte tells the bootloader how the EEPROM is structured and the sections are used as follows:

Table 3–5: Header Byte

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1=test checksum	0	0	0	1=Section4	1=Section3	1=Section2	1=Section1

Section1: default data and programmable descriptor components (see The Section1 Concept)

Section2: used for plug-ins; small pieces of code that extend a given ROM firmware

Section3: contains the V8 code for the Shadow RAM

Section4: contains the XDFP code; the bootloader loads it into the external DSP Emu-RAM

3.1.6.1. Content of the external EEPROM

Table 3–6:

Addr.	Content	Byte
0x00	length in bytes of Section1 + 1 for checksum	High Byte
0x01	length in bytes of Section1 + 1 for checksum	Low Byte
0x02	address-offset in bytes of Section1 related to addr 0000	High Byte
0x03	length in bytes of Section2 + 1 for checksum	High Byte
0x04	length in bytes of Section2 + 1 for checksum	Low Byte
0x05	address-offset in bytes of Section2 related to addr 0000	High Byte
0x06	length in bytes of Section3 + 1 for checksum	High Byte
0x07	length in bytes of Section3 + 1 for checksum	Low Byte
0x08	address-offset in bytes of Section3 related to addr 0000	High Byte
0x09	length in bytes of Section4 + 1 for checksum	High Byte
0x0A	length in bytes of Section4 + 1 for checksum	Low Byte
0x0B	address-offset in bytes of Section4 related to addr 0000	High Byte
0x0C	address-offset in bytes of Section4 related to addr 0000	Low Byte
0x0D	DATA of Section1 + 1 for checksum	Header Byte
0x0E	...	
0xxx	DATA of Section2 + 1 for checksum	Header Byte
0xxx	...	
0xxx	DATA of Section3 + 1 for checksum	Header Byte
0xxx	...	
0xxx	DATA of Section4 + 1 for checksum	Header Byte
0xxx	...	

The checksum is an XOR over all bytes of a section the address-offset of Section1 is 0x11 (17 Bytes Header)

3.1.7. The Plug-in Concept

4. USB Descriptor

4.1. USB_DESCRIPTOR.ASM

This part of the source code covers all information that is reported to the host upon enumeration. The „DESCRIPTOR“ provides a complete description of the USB-characteristic. It is assumed that the reader is familiar with the USB-basics and it is recommended to have the USB-SPEC 1.1 nearby because it is often referred to this paper. The following paragraphs explain the structure of the descriptor and the functionality of the Micronas Standard Descriptor.

The descriptor is split into the following parts:

– Device Descriptor
– Configuration Descriptor
– Interface Descriptors

4.1.1. Device Descriptor (Chapter 9.6.1)

1here is the place to provide formal information of the device like

– Manufacturer ID
– Product ID
– Release Numbers
– String Information

The Micronas Standard Descriptor keeps this kind of information EEPROM programmable (->FE-concept, ->Section1-concept). This, however, is important only when the device works with ROM-Firmware and taking only few programmable parts out of an external EEPROM.

4.1.2. Configuration Descriptor (Chapter 9.6.2)

Sometimes all but the device descriptor is called Configuration Descriptor but the exact definition is given in chapter 9.6.2 of the spec. The important points here are:

– number of interfaces (Chapter 9.6.3, note: wrong number here crashes all operating systems!)
– Attributes - defines self/bus-powered and remote wake-up capability
– Maxpower in mA - this must be set in case of bus-powered mode

The attributes and the maxpower are programmable via Section1.

The Micronas Standard Firmware uses the configuration string for a time code information

It is string 3 (index=3), and the V8-assembler directive „..datetime“ generates the timing information automatically. This allows an easy check of the FW-version.

4.1.3. Interface Descriptors (Chapter 9.6.3)

All USB-functions require a USB interface. The Standard Descriptor has 4 interfaces. This is coded as follows:

```
.equ    bNumInterfaces4
.equ    ISO_CONTROL_IFC_NR0
.equ    ISO_DOWN_IFC_NR1
.equ    ISO_UP_IFC_NR2
.equ    HID_IFC_NR3
```

The **ISO_CONTROL_IFC** is a class specific descriptor and describes the path of the up- and downstream audio signal to and from the DSP. Here we define whether the device is an USB-Speaker, USB-Headset, or a USB soundcard for example. We also define here, how the device appears in the Windows mixer, for example if we have line-input or mic input or both. The rules for defining this structures can be found in the „Universal Serial Bus Device Class Definition for Audio Devices“.

Note: The USB-Spec allows free combinations of input units, feature units, selector units, mixer unit, output units etc. A first serious restriction here, however, is the Windows mixer, which must be observed, if the device shall be shipped without additional drivers. The WIN mixer is not flexible and requires a strict scheme of devices working on the recording mixer and devices working on the playback mixer. Also the choice of features is limited here. For example: we don’t have mute buttons for the recordings devices, or how shall we switch on and off an AGC for the mic-input? No way.

Apart from that, the WIN-mixer shows different performance in different WIN-versions. The mixer WIN-ME is nearly unusable.
The second restriction is the limited capability of the operating system in parsing this part of the descriptor due to a bad implementation of the USB-Spec. This causes many blue-screens or system hang-ups and requires try & error methods, because nobody can give a reliable answer here on how the descriptor should look like. The Micronas Standard Descriptor was tested on all WIN – operating systems and on MAC 9.1.

The **ISO_CONTROL** interface reflects the following audio path in the Micronas Standard Descriptor.

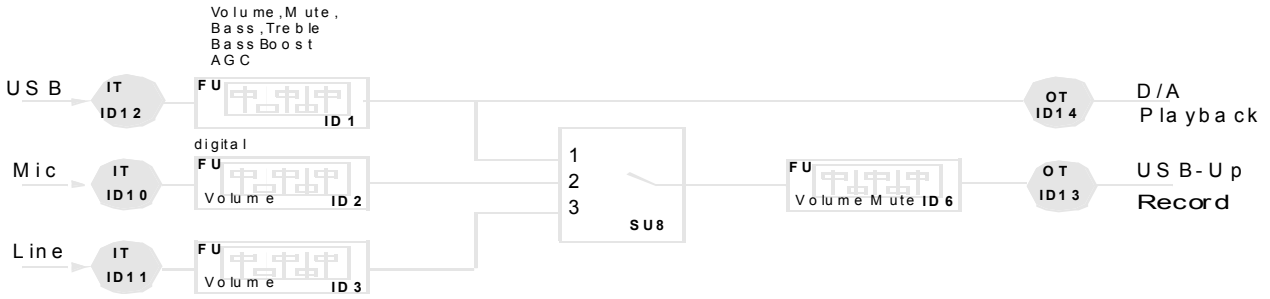

Fig. 4–1:

Its the basic configuration of an USB codec. The additional (and already implemented) asynchronous I2S I/O features are not reflected here.

Other more complex structures are possible, but then the operating system eventually may not handle this and an additional vendor specific driver and mixer tool may be required.

The **ISO_DOWN interface** covers the audio-downstream functionality. This interface has two alternate settings, describing the audio formats, which are supported by the chip. These are:

Alternate Setting 1	16 bit mono
Alternate Setting 2	16 bit stereo

The properties of each alternate setting are defined in the

CS-Audio Streaming Interface Format. Here it is declared that the chip handles continuos sampling rates. The range is defined with:

;ISO-SAMPLING RANGE

.equ LOW_SAMPLE_FREQ 6400

.equ HI_SAMPLE_FRE48000

This range is identical for all ISO-interfaces. The upper limit must never be exceeded.

Important also is the packet size which define the maximum bytes per frame. It is recommended to slightly exceed the theoretical value.

Example:16_BIT_STEREO at 48kHz require 48 x 2 x 2 = 192bytes

the value used in the descriptor is 200bytes

The following packet sizes should never be changed.

;PACKET SIZE

.equ 8_BIT_MONO 50

.equ 8_BIT_STEREO100

.equ 16_BIT_MONO100

.equ 16_BIT_STEREO200

.equ 24_BIT_MONO150

.equ 24_BIT_STEREO300

Note: The device is capable of handling 24bit downstream word also. But the standard descriptor does not declare this. The code for this (alternate setting 3), however, is provided.

The downstream uses endpoint 1.

The **ISO_UP interface** covers the audio-upstream functionality. This interface has three alternate settings, describing the audio formats, which are supported by the chip. These are:

Alternate Setting 1	08 bit mono
Alternate Setting 2	16 bit mono
Alternate Setting 3	16 bit stereo

The upstream uses endpoint 4 (coded as 84 – IN endpoint)

The HID interface contains the HID endpoint 3 (coded as 83 – IN endpoint) and the REPORT Descriptor. The Standard Descriptor assigns the following functions to the GPI-Ports:

GPI 0	Volume Up
GPI 2	Volume Down
GPI 3	Mute
GPI 4	Bass Boost

The key codes for GPIO5..7 are also reported to the host, but no HID-functions are assigned to this pins. This can be used in a vendor specific driver to trigger additional functions.

Important for the HID endpoint are:

– 0x0002;wMaxPacketSize

– .byte0xfe;bInterval

bInterval defines the polling interval in ms. This is a Section 1 variable (->FE-concept, ->Section1-concept).

Note: It is not recommended to change the MaxPacketSize in the HID-endpoint, because this requires deep familiarity with the TX-buffer handling.

More information on the HID handling can be found in the description of gpio_control.asm.

5. Handling of Class Specific Requests

5.1. Introduction

The UACB needs class specific requests to handle the USB audio control. After an incoming request is identified as class specific further processing is done in the USB_CLASS_SETUP routine, located in usb_class.asm.

Here first it is checked if its an audio class request. This happens in USB_AUDIO_CTL_START in the file usb_audio_ctl.asm. If it is not identified here the same is tried with USB_HID_CTL_START and USB_VEND_CLAS_START but only audio-control and sample rate-control is supported in the standard firmware

5.2. Audio-Control Handling (usb_audio_ctl.asm, usb_audio_ctl_include.asm, dsp_control.asm)

It is recommended here to be familiar with the Universal Serial Bus Device CLass Definition for Audio Devices Specification, Release 1.0.

The number and kind of control request is strictly related to the **ISO_CONTROL_IFC** descriptor. That means if the descriptor is modified also the request handling needs to be modified (especially if new features are introduced).

The standard firmware handles the following audio control requests for feature units (FU), selector units (SU) and mixer units (MU).

- FU1_MUTE_
- FU1_VOL_CH1
- FU1_VOL_CH2
- FU1_BASS
- FU1_TREBLE
- FU1_AGC
- FU1_BASSBOOST
- FU2_MUTE
- FU2_VOL
- FU3_MUTE
- FU3_VOL
- FU6_MUTE
- FU6_VOL
- SU7
- SU8
- MU9_IN1
- MU9_IN2

This was based on a descriptor which covered more functionality than the current descriptor.

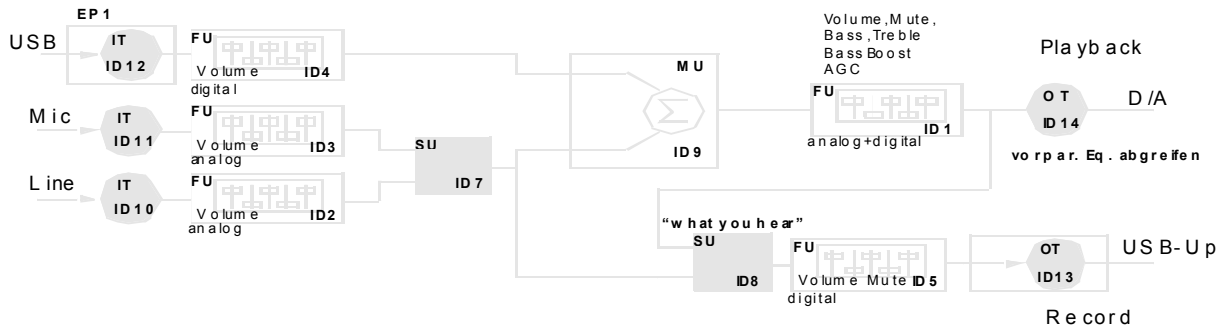

Fig. 5–1:

Note: Even this structure does not represent the complete audio structure of the device. All I2S functionality is not represented here and needs to be added in a future version.

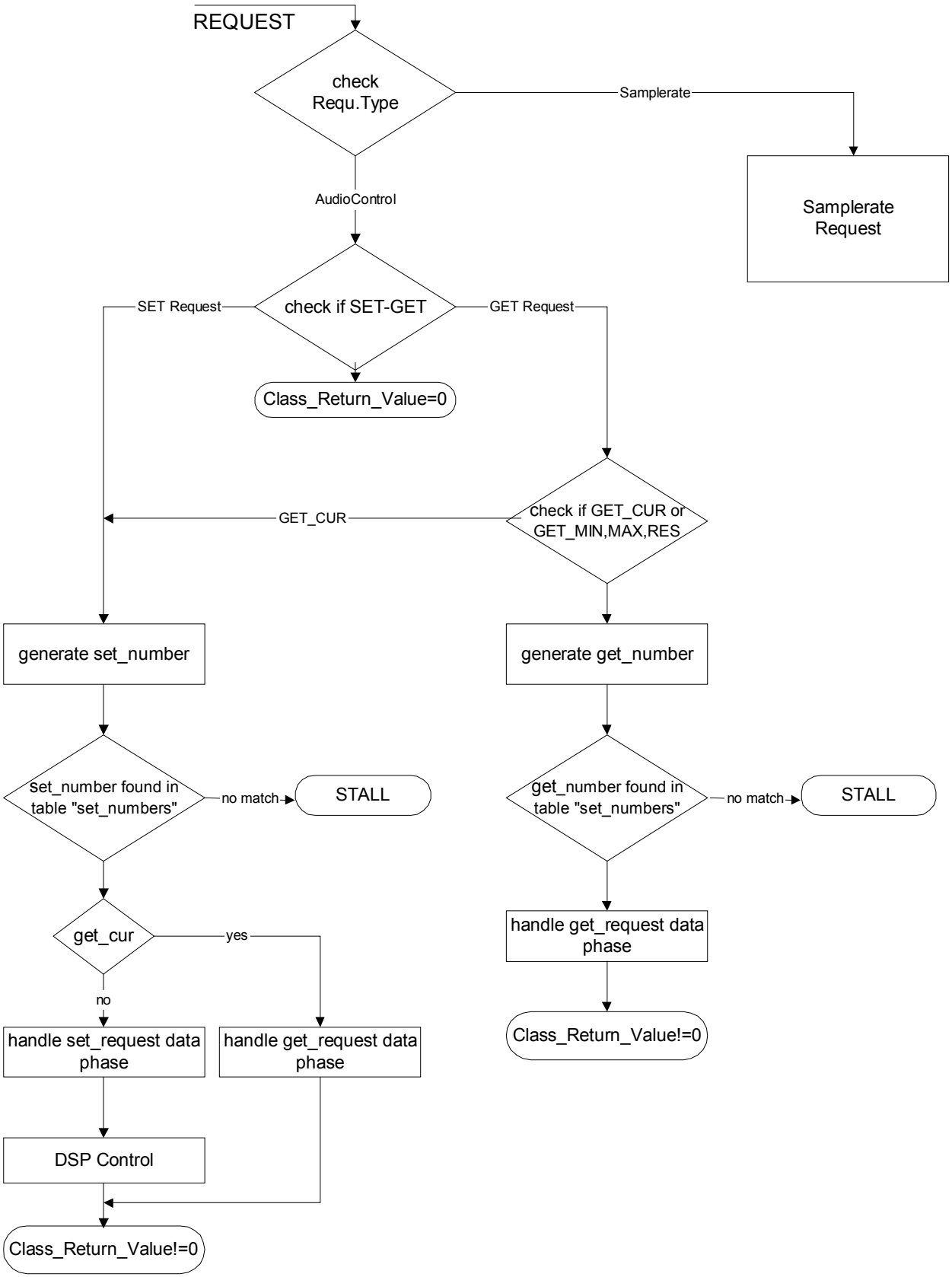

Fig. 5–2: Audio request handling

5.2.1. GET-Requests and SET-Requests.

The GET-requests are needed

1. to report the parameter range (MIN,MAX,RES) of the units to the host.

   This information is located in ROM-table ROM_USB_MINMAXRES in usb_audio_ctl_include.asm

   ROM_USB_MINMAXRES:

   ROM_USB_VOL_MIN_FU1_CH1: .rword 0xD700

   ROM_USB_VOL_MAX_FU1_CH1: .rword 0x0000

   ROM_USB_VOL_RES_FU1_CH1: .rword 0x0010

   etc.

   This, for example, tells the host that FU1 has an audio range from –40dB to 0dB with a stepsize of 1dB.

   All parameters here are coded with 2-bytes even if only 1-byte is required. This simplifies programming

2. to report the current status (CUR) of the unit.

   This information is located in RAM-table USB_AUDIOCLASS_SET_CUR in usb_audio_ctl_include.asm. The content of this table represents the current status of the audio parameters.

   The CUR-parameters are not exactly identical with the values which are transmitted to the DSP.In some cases there must be a transformation first, to match the requirements of the DSP - algorithm.

The SET requests are needed:

1. to set the CUR-parameters
   This request modifies the RAM-table USB_AUDIOCLASS_SET_CUR and accordingly triggers the update of the DSP-parameters (dsp_control.asm)

2. to set the sampling rate for the ISO – upstream (endpoint control operation)

The method used for decoding the request types is table based. The idea behind this is:

The supported requests are represented by a certain number, which is calculated out of the requests components. These „magic numbers“ are stored in the ROM-tables „set_numbers“ and „get_numbers“. Out of any incoming vendor request the same procedure is used to compute this number and then it is checked if this number is found in the set_numbers or get_numbers table. If the number is found, the corresponding table position represents the decoded request.

If more requests needs to be decoded than provided in the standard firmware the procedure is needed to compute the „magic numbers“. This is done with the help of Table 5–1 an Table 5–2.

:

Table 5–1: SET numbers

Request Nr.	wIndexH	wValueH		wValueL	magic number	Request Name
1	1	1		0	16	FU1_MUTE_CUR
2	1	2		1	288	FU1_VOL_CH1_CUR
3	1	2		2	544	FU1_VOL_CH2_CUR
4	1	3		0	48	FU1_BASS_CUR
5	1	5		0	80	FU1_TREBLE_CUR
6	1	7		0	112	FU1_AGC_CUR
7	1	9		0	144	FU1_BASSBOOST_CUR
8	2	1		0	17	FU2_MUTE_CUR
9	2	2		0	33	FU2_VOL_CUR
10	3	1		0	18	FU3_MUTE_CUR
11	3	2		0	34	FU3_VOL_CUR
12	6	1		0	21	FU6_MUTE_CUR
13	6	2		0	37	FU6_VOL_CUR
14	7	0		0	6	SU7_CUR
15	8	0		0	7	SU8_CUR
16	9	1		1	280	MU9_IN1_CUR
17	9	2		1	296	MU9_IN2_CUR

Table 5–2: GET numbers

Request Nr.	wIndexH	wValueH	wValueL	bRequest	magic number	Request Name
1	1	2	1	82	288	USB_VOL_MIN_FU1_CH1
2	1	2	1	83	1312	USB_VOL_MAX_FU1_CH1
3	1	2	1	84	2336	USB_VOL_RES_FU1_CH1
4	1	2	2	82	544	USB_VOL_MIN_FU1_CH2
5	1	2	2	83	1568	USB_VOL_MAX_FU1_CH2
6	1	2	2	84	2592	USB_VOL_RES_FU1_CH2
7	1	3	0	82	48	USB_BASS_MIN_FU1_CH0
8	1	3	0	83	1072	USB_BASS_MAX_FU1_CH0
9	1	3	0	84	2096	USB_BASS_RES_FU1_CH0
10	1	5	0	82	80	USB_TREBLE_MIN_FU1_CH0
11	1	5	0	83	1104	USB_TREBLE_MAX_FU1_CH0
12	1	5	0	84	2128	USB_TREBLE_RES_FU1_CH0
13	2	2	0	82	33	USB_VOL_MIN_FU02_CH0
14	2	2	0	83	1057	USB_VOL_MAX_FU02_CH0
15	2	2	0	84	2081	USB_VOL_RES_FU02_CH0
16	3	2	0	82	34	USB_VOL_MIN_FU03_CH0
17	3	2	0	83	1058	USB_VOL_MAX_FU03_CH0
18	3	2	0	84	2082	USB_VOL_RES_FU03_CH0
19	6	2	0	82	37	USB_VOL_MIN_FU06_CH0
20	6	2	0	83	1061	USB_VOL_MAX_FU06_CH0
21	6	2	0	84	2085	USB_VOL_RES_FU06_CH0
22	7	0	0	82	6	USB_MIN_SU07
23	7	0	0	83	1030	USB_MAX_SU07
24	7	0	0	84	2054	USB_RES_SU07
25	8	0	0	82	7	USB_MIN_SU08
26	8	0	0	83	1031	USB_MAX_SU08
27	8	0	0	84	2055	USB_RES_SU08
28	9	1	1	82	280	USB_MIN_MU09_IN1
29	9	1	1	83	1304	USB_MAX_MU09_IN1
30	9	1	1	84	2328	USB_RES_MU09_IN1
31	9	2	1	82	296	USB_MIN_MU09_IN2
32	9	2	1	83	1320	USB_MAX_MU09_IN2
33	9	2	1	84	2344	USB_RES_MU09_IN2

The set numbers are computed with:

(wIndexH-1) + wValueH*16 + wValueL*256

The get number are computed with

=($B23-1)+$C23*16+($E23-82)*16*16*4+$D23*16*16

wIndexH-1 + wValueH*16 + bRequest*256 + (wValueL-82)*1024

Both set and get numbers are represented with 16bit which is sufficient for this purpose. The other request components are checked by normal „if“-structures.

The flowchart should give a better understandig of this procedure.

Note: The GET_CUR requests also uses the set_number table, because the structure of these request is very similiar to the SET_CUR requests.

Note: The SET_INDEX is used to identify the request. It the table position where the magic number matches the table entry. In the dsp_control.asm this index is used as a „jump handler“ to the routines which transfer the audio control parameters to the DSP.

5.2.2. Examples

An example will illustrate how a new feature is added.

5.2.2.1. How to add new Features and Requests

We want to add the loudness function as an addtional tone control in the audio path. This is already implemented in the DSP and only needs to be activated. In order to do that we need the following steps.

How to change the Descriptor

We need to modify the feature unit 1 (FU1) in the following way:

```
        .equ            FU_ID1     1
;CS-specific Feature Unit Descriptor
        .byte           0x0d            ;bLength
        .byte           0x24            ;Audio class specific interface desc
        .byte           0x06            ;Feature Unit descriptor
AUDIO_FEATURE_UNIT_ID1:
        .byte           FU_ID1          ;Unit ID
        .byte           MU_ID9          ;Source ID
        .byte           IT_ID12         ;Source ID  ###KS
        .byte           0x02            ;ControlSize Length in bytes of bmaControls
        .rword          0x0355          ;bmaControls(0);defines supported features in master channel
        .rword          0x0002          ;bmaControls(1);defines supported features in log. channel 0
        .rword          0x0002          ;bmaControls(2);defines supported features in log. channel 1
        .byte           0x00            ;iFeatureindex of string descriptor, descr. feature unit
```

Loudness works on both channels, so we need to modify the bma Control(0), which lists all feature in the master channel (all features which cannot controlled separetely for left and right channel are listed here). Looking into the audio class spec tells us that loudness is activated by bit9 in the bma-Control(0). So the new bmaControl is:

```
.rword  0x0355                  ;bmaControls(0)
```

How to implement the Request Handling

Upon enumeration the host now knows that the device supports loudness, and tries to read the characteristics of this feature. The only parameter of interest here is a boolean variable, that switches loudness on and off. Unlike volume requests or bass/treble request there are no MIN/MAX/RES parameters here. So we only have to extend the SET-table.

We add Request Nr. 18:

Request Nr.	wIndexH	wValueH		wValueL	magic number	Request Name
18	1	10		0	296	FU1_LOUDN_CUR

The magic number according to the formula:

(wIndexH-1) + wValueH*16 + wValueL*256 is then

0 + 10x16 + 0x256 = 160

This number is added at the end of the set-table

```
...
        .rword      280     ; MU9_IN1_CUR
        .rword      296     ; MU9_IN2_CUR
        .rword      160     ; FU1_LOUDN_CUR
```

This allows to detect if the host wants to set or get the CUR-value.

...

Now we need to add a new RAM-variable **USB_FU1_LOUDN_CUR,** which holds the CUR-status:

```
SB_SU8_CUR:          .rword     0x0000    ;15
USB_MU9_IN1_CUR:     .rword     0x0000    ;16
USB_MU9_IN2_CUR:     .rword     0x0000    ;17
USB_FU1_LOUDN_CUR:   .rword     0x0000    ;18
```

Remember: the set-table and the CUR-table are both located in usb_audio_ctl_include.asm

How to implement the DSP-control

From the USB-host point of view everything is ok now: the device reports loudness as an audio feature, the device accepts setting the CUR-value, and the device reports the CUR-value. But there is still one action missing: the V8 needs to switch the loudness function on and off in the DSP.

```
....
jeq     XDFP_RECORD_SELECT;14
dec     r0
jeq     XDFP_ANALOG_VOLUME;15
dec     r0
jeq     XDFP_USBDOWN_VOLUME;16
dec     r0
jeq     XDFP_LOUDNESS
rts
```

and then, of course, we need to add the new routine XDFP_LOUDNESS. The easiest way to do this is to copy and paste an already existing function and change some names

```
XDFP_LOUDNESS:
.equ    XDFPADR_PLAYBACK_LOUDN0x15   ; XDFP-ADDRESS for Loudness
.equ    XDFP_LOUDNESS0x30
; XDFP needs 0x00=Loudness-OFF       ; 0x30=Loudness-ON
; USB sends 0=Loudness-OFF 1=Loudness-ON
ldi     r2, XDFP_LOUDNESS            ; this defines the Loudness-Characteristic
lda     r1, USB_FU1_LOUDN_CUR
brne    LOUDNON
lda     r2, 0x00                     ; Loudness off
LOUDNON:
ldi     r1, XDFPADR_PLAYBACK_LOUDN
jmp     XDFP_SEND_3BYTES             ;sends Loudness,0,0 (Hi,Mi,Lo) with rts
```

Another option here would be to put the XDFP_LOUDNESS into the Section1 as a parameter that comes from the EEPROM, because the loudness characteristic in the DSP can be set to various modes. Using 0x30 is just one example. See the document UACB – DSP firmware for more info on the loudness.

This happens in dsp_control.asm. This routine is called after the SET-request and uses the RAM-variable SET_INDEX as branching control to the corresponding DSP-control-routines. The SET_INDEX is the SET-table position of the detected request, i.e. 18 for our new request.

We extend the branch structure in dsp_control.asm

Initialisation

The DSP sets all variables to zero after power up. This means for our example that loudness is off. This, however has to correlate with the CUR-value in the V8. Without any initialisation the CUR-values are all set to zero after power up and this is exactly what we need. But it is recommended to always do the initialisation. This is done in the main_include.asm.

```
...
;       init the Audio_Class

        ldi r0,         0x80
        sta r0, USB_FU1_VOL_CH1_CUR     ;mute the playback volume
        sta r0, USB_FU1_VOL_CH2_CUR     ;mute the playback volume

        ldi r0, 0x01
        sta r0, USB_SU7_CUR             ; init the SU7 to Mic-Input
        xor r0
        sta r0, USB_FU1_LOUDN_CUR
```

Now everything is ready to switch Loudness on and off under control of the operating system. Unfortunately the WIN mixer does not provide any controls to do that, so this feature was not implemented in the standard firmware.

Fig. 5–3:

5.3. DSP Control

The code in dsp_control.asm handles the communication between the V8 and the DSP. In the standard firmware its is a one-way communication, which means, data are sent to from the V8 to the DSP, but nothing is read back. Other applications, however, may require this, and therefore the hardware is prepared to do that.

This hardware is highlighted first because it is essential for understanding this communication.

The next figure shows the control interface between V8 and DSP:

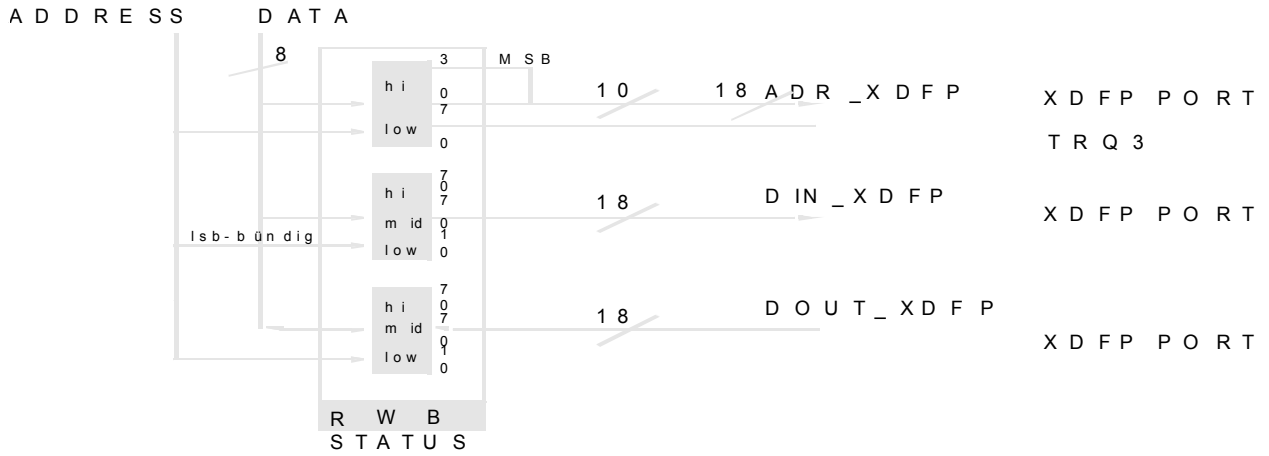

Fig. 5–4:

The DSP-RAM has a wordlength of 18-bit and so we need 3-bytes from the V8 to completely write one DSP-location. So we have 3-bytes both for each direction. In case of writing into the DSP these bytes are named:

– XDFP_DATAIN_HI

– XDFP_DATAIN_MI

– XDFP_DATAIN_LO

and this bytes represent one DSP word as follows:

XDFP_DATAIN_	HI-7	HI-6	HI-5	HI-4	HI-3	HI-2	HI-1	HI-0	MI-7	MI-6	MI-5	MI-4	MI-3	MI-2	MI-1	MI-0	LO-1	LO-0
DSP-word	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The address space of the DSP is 10-bit, so we need 2-bytes to address this. This bytes are named:

– XDFP_ADR_HI

– XDFP_ADR_LO

and build the DSP_address plus R/W -control as follows:

XDFP_ADR_	_HI7	_HI6	_HI5	_HI4	_HI3	_HI2	_HI1	_HI0	_LO7	_LO6	_LO5	_LO4	_LO3	_LO2	_LO1	_LO0
DSP_ ADDRESS						0= RD 1=WR	9	8	7	6	5	4	3	2	1	0

The Read, Write, Busy status-bits are needed as handshake signals. Their meaning will become clear after the description of the write and read sequence below:

Write Sequence:

V8 has to transmit:

– XDFP_DATAIN_LO

– XDFP_DATAIN_MI

– XDFP_DATAIN_HI

– XDFP_ADR_HI

– XDFP_ADR_LO

The transmission of the low-addressbyte generates the write-interrupt for the DSP and sets the W- and B-bit in the statusregister.
The W- and the B-Bit are reset, when the DSP reads the in-data-register

The W-bit is redundant here and is not used in the standard firmware

Read Sequence

V8 has to transmit:

– XDFP_ADR_HI

– XDFP_ADR_LO

The transmission of the low-addressbyte generates the read-interrupt for the XDFP and sets the R- and the B-bit. The DSP reads the address and writes the required data word into the out-data register This resets the R-bit.

V8 has to poll the R-bit and fetch the 3-bytes after the R-bit is reset:

– XDFP_DATAOUT_HI

– XDFP_DATAOUT_MI

– XDFP_DATAOUT_LO

The B-bit is reset after the LO-data is fetched from the V8.

The write-sequence is already implemented in the standard firmware and there should be no need to change this. The routine is located in the upper-rom and is called **XDFP_SEND_3bytes**. This routine handles the complete write sequence, including the handshake (using B-bit).

The values are passed by registers:

– r1 = XDFP_ADR_LO

– r2 = XDFP_DATAIN_HI

– r3 = XDFP_DATAIN_MI

– r4 = XDFP_DATAIN_LO

There is no XDFP_ADR_HI and so only the lower 256 words within the DSP-RAM-space can be addressed with this routine. The DSP-firmware, however, holds all control variables within this lower address space, so there is no need for the HI-byte.

In the same way a **XDFP_READ_3bytes** can be created.

5.4. Sample Rate Control

The samplerate requests are endpoint control requests and are needed for ISO upstreaming.The SET_SAMPLERATE tells the device how many audio samples per frame the host expects.The samplerate is represented as a 3-byte word, but for the UACB samplerate range (6.4kHz ...48kHz) only 2-bytes are required, so the high byte is ignored for further processing.

The samplingrate is specified in Hz. So 0x00bb80 is transferred with the SET_SAMPLERATE request, when 48000Hz is required. This number is the input for an algorithm that computes the number of samples per frame, i.e. samples per millisecond, what is basically a „devide by 1000“ operation in case of „integer multiples of 1000“ - samplerates. In all other cases the „devide by 1000“ algorithm result in a nonzero remainder. This remainder is then accumulated 1000 times and when the accumulator overflows an additional sample is transmitted.

The samplerate goes into the DSP too. Here it is needed as an initial value for the samplerate conversion.

Note: The A/D conversion is running on 48kHz, independantly of what the iso-upstream requires. The adaption to the required samplerate is done by samplerate conversion.

Note: It is observed that SET_SAMPLERATE is also transferred preceeding an iso-downstream. This was not the case with downstream-only devices (UACA). Here this information is not required but on the other hand it is not allowed to stall this request. So it is handled also. Otherwise the host would never send iso-downstream data.

The following samplingrate request are supported:

– SET CUR

– GET CUR

– GET MIN

– GET MAX

– GET RES

After a SET CUR request the samples per frame are calculated and stored in the upstream BDT.

At this point the complete algorithm is running including the „devide by 1000“ code („subtract 1000 until result is negative“).

During ISO-uopstream only the fractional part of the „samples per frame“ algorithm is computed. The integer part does not change at a given samplerate.

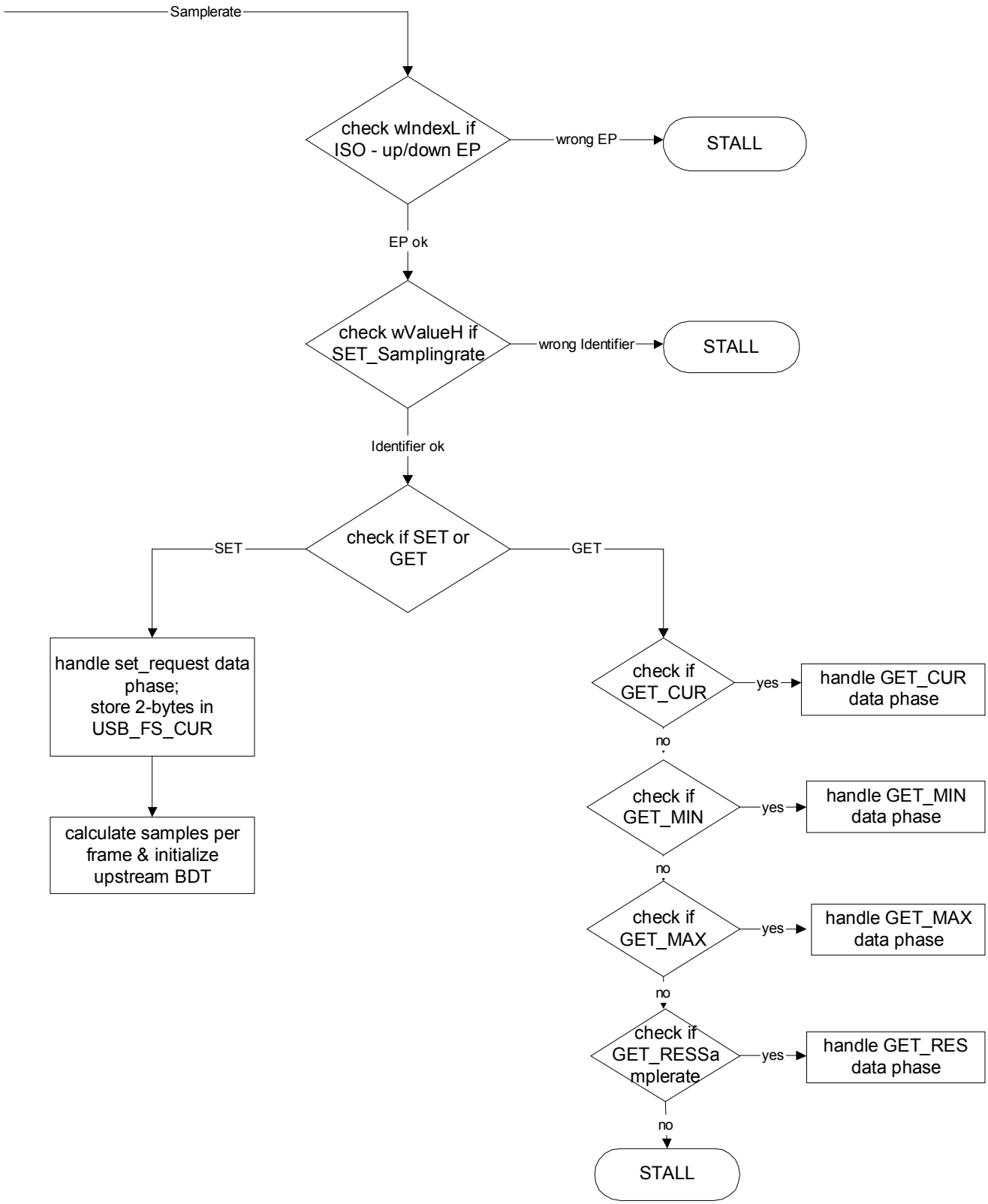

Fig. 5–5: Sample rate control handling

6. GPIO Handling

This chapter describes the usage of the 12 GPIO pins. First a description of the various modes is given.

Each GPIO-pin can be used either as input or as output pin. The direction of each pin is defined in the following hardware control registers:

_LO defines GPIO0....7LSB == GPIO0, MSB == GPIO7

_HI defines GPIO8..11LSB == GPIO8; BIT3 == GPIO11

GPIO_CONFIG_LO0xb0510 = input; 1 = output

GPIO_CONFIG_HI0xb058

In case of output, the padstrength (value???) can be set to either high or low:
(for other pins see the UACB – V8 reference)

GPIO_PADSTRENGTH_LO:0xb0550 = weak; 1 = strong

GPIO_PADSTRENGTH_HI:0xb056

In case of input, an internal pulldown (250k) can be switched either on and off

GPIO_PULLDOWN_LO0xb0590 = pulld. on; 1 = pulld. off

GPIO_PULLDOWN_HI0xb05a

The standard firmware defines the default GPIO characteristic in Section1:

```
SED_DEFAULT_GPIO_CONFIG_LO_DEFAULT:; default 0, 1 = output
    .byte 0x00
SED_DEFAULT_GPIO_CONFIG_HI_DEFAULT:; default 0, 1 = output
    .byte 0x00
SED_DEFAULT_GPIO_PULLDOWN_LO_DEFAULT:; default: 1 (pulldown on)
    .byte 0xff
SED_DEFAULT_GPIO_PULLDOWN_HI_DEFAULT:; default: 1 (pulldown on)
    .byte 0xff
SED_DEFAULT_GPIO_PADSTRENGTH_LO_DEFAULT:; default: 0 (weak)
    .byte 0x00
SED_DEFAULT_GPIO_PADSTRENGTH_HI_DEFAULT:; default: 0 (weak)
    .byte 0x00
```

This values are copied from the bootloader into the Section1 RAM mirror image (see Section1 concept) and from there copied into RAM shadow registers. This was required because the hardware control register are read only and so the V8 needs to keep track of the actual status either for internal purpose or for reporting it to an external controller (USB-Host, I2C-Controller etc). This is used for some other hardware register too. Unfortunately this wastes some RAM space

The following sequence (including the RAM addresses) can be found in main_include.asm:

```
A709  lda r0, GPIO_CONFIG_LO_DEFAULT
A184  sta r0, GPIO_CONFIG_LO_SHADOW
A70A  lda r0, GPIO_CONFIG_HI_DEFAULT
A185  sta r0, GPIO_CONFIG_HI_SHADOW
A70B  lda r0, GPIO_PULLDOWN_LO_DEFAULT
A186  sta r0, GPIO_PULLDOWN_LO_SHADOW
A70C  lda r0, GPIO_PULLDOWN_HI_DEFAULT
A187  sta r0, GPIO_PULLDOWN_HI_SHADOW
A70D  lda r0, GPIO_PADSTRENGTH_LO_DEFAULT
A188  sta r0, GPIO_PADSTRENGTH_LO_SHADOW
A70E  lda r0, GPIO_PADSTRENGTH_HI_DEFAULT
A189  sta r0, GPIO_PADSTRENGTH_HI_SHADOW
```

The shadow registers are then copied into the hardware register in the routine V8_RESTORE_SHADOW_REGS (also located in main_include.asm). The routine V8_RESTORE_SHADOW_REGS runs after RESET and is also called in the scheduler taks 5 (see „The Scheduler Concept“). By setting the associated scheduler bit the routine runs every 1ms.

This automatic update allows to control the GPIOs just by setting the shadow registers. In this case, however, it must be observed that there may be a delay of up to 1ms between writing into the shadow registers and the update of the hardware register. If this delay is not acceptable, the hardware registers needs to be controlled directly. In this case, however, it is strictly recommended to update the shadow registers too, in order to have identical information in both hardware and shadow registers.

In most applications the configuration of the GPIOs will be static, i.e: a pin is either input or output, with or without pulldown etc., and this configuration is defined in section1 as default data.

More important for applications, however, is the writing or reading to or from the GPIO pins itself.

The status of the lower bits 0...7 is defined in:

b0A0GPO_REG[0:7]

The status of the upper bits 8...11 is defined in:

b0A1GPIO_REG[4...7]

The status of the lower input pins 0...7 is found in:

b0A2GPI_REG[0...7]

The status of the upper input pins 8...11 is found in

b0A1GPIO_REG[0...3]

Here we don’t need the shadow register concept, because all this register are readable.

Note: The standard firmware uses only GPIO0..3 as input pins for the HID functions.

6.1. Memory Mapped Mode

The GPIOs can also be used in a memory mapped mode. This means the address range

b0b0...b0bf

is transparent to the GPIOs. GPIO0..7 are mapped to the databus and GPIO8...11 are mapped to the lower four bits of the addressbus. This means for example, writing a 0xaa to b0bf results in 0xaa at the GPIO0...7 and 0xf at GPIO8...11. In this mode the GPIO8...11 need to be set as output pins, of course.

How does the standard firmware use the GPIOs?

Looking into the report descriptor tells that four HID audio control functions are defined:

```
; assigned real keys
    .rword      0xE909      ; Usage Volume UP
    .rword      0xEA09      ; Usage Volume DOWN
    .rword      0xE209      ; Usage MUTE
    .rword      0xE509      ; Usage BASS BOOST
```

This functions are assigned to the GPIO0...3. High level at GPIO0 ramps up the volume for example.

This means this pins need to be polled by the V8, in order to report their state on request to the host.

As polling rate the 1ms timebase is used, which is provided in the scheduler. Scheduler task 0 calls the routine USB_GET_KEY located in gpio_control.asm. Some notes on the USB_GET_KEY is given below.

6.2. Debouncing

For debouncing a simple timer function is used. After detection of high level at one of the GPIO..3 a timer (KEY_TIMER) is started (software timer) counting from MAX_KEY_TIMER down to zero. MAX_KEY_TIMER is defined in Section1

```
SED_DEFAULT_MAX_KEY_TIMER:
    .byte 0x7f
```

As long as the counter did not finish zero, there is no new polling. This timer also defines the repetition rate of a permanent pressed key (defines for example, how fast the volume slider in the WIN mixer goes up and down).

Note: This simple method does not protect from noise or spikes on the pins. For this kind of problems averaging algorithm should be used.

6.3. Reporting the Keycode

according to the HID endpoint descriptor the USB-host sends IN-tokens to the EP3 at a fixed rate which is defined in the

.byte 0xfe;bInterval

variable. The standard firmware holds this value as a Section1 variable (see „The FE-concept“)

.from data_sec1.asm.

SED_DEFAULT_USB_CONFIG_ATTRIBUTES_CUSTOM:

```
    .byte       0xc0        ; bmAttributesattributes - self powered
    .byte       0x00        ; MaxPowerself-powered draws 0 mA from the bus.
    .byte       0x08        ; bInterval, TR 7.7.99 was the *last* byte of the config_descriptor
```

0x08 means, that the IN-token comes every 8ms or in every 8th frame.

The IN-token expects to get the number of the key, which may be pressed, as a so called keycode in the data phase. So, for example, if high-level at GPIO3 is dedected, a 0x0003 is reported after the next IN-token. The USB-host associates this keycode with the HID function MUTE-toggle. The standard firmware uses 2-bytes in the data phase.

Note: Also the unassigned keys are reported (GPIO4...7) and can be used for vendor specific purpose.

```
;;HID Endpoint Descriptor
    .byte0x07;bLength
    .byte0x05;bDescriptortype = Endpoint
    .byte0x83;bEndpointAddress
    .byte0x03;bAttributes
    .rword0x0002;wMaxPacketSize;db 29.09.1999 0305 because of ext. HID
    .byte0xfe;bInterval TR 7.7.99 moved to EEPROM
```

6.4. Toggle Keys

GPIO2 (mute) and GPIO3 (bassboost) are used as toggle keys. This means, if this keys keep pressed all the time, it is not wanted that the assigned function toggles between on and off. The state should just toggle once and stay there until the key is released and pressed again. Although there may be a smarter way to do it, this is solved by sending a „keycode without function“ after the keytimer accepts a new polling. Without this the same keycode would be sent all the time.

6.5. GPIO port configurations.

The UAC 357xB can set the GPIO port into different configurations.

– Standard mode

– Address mode

The GPIOs can be grouped in 2 different types:

– Input and output pins: GPIO[0..11]

– Control pins: RD, STRB

The port pins can also be set into different electrical states:

– weak or strong driver strength

– output or tristate

– internal pulldown on or off

Table 6–1: GPIO port configurations

Pin name	Standard mode	Address mode
GPIO [0]	– Generic I/O – HID: Volume down	Generic parallel I/O
GPIO [1]	– Generic I/O – HID: Volume up	Generic parallel I/O
GPIO [2]	– Generic I/O – HID: Mute button	Generic parallel I/O
GPIO [3]	– Generic I/O – HID: Bass boost button	Generic parallel I/O
GPIO [4]	Generic I/O	Generic parallel I/O
GPIO [5]	Generic I/O	Generic parallel I/O
GPIO [6]	Generic I/O	Generic parallel I/O
GPIO [7]	Generic I/O	Generic parallel I/O
GPIO [8]	Generic I/O	Adr [0]
GPIO [9]	Generic I/O	Adr [1]
GPIO [10]	– Generic I/O – Start timer	Adr [2]
GPIO [11]	– Generic I/O – PWM out	Adr [3]
RD	no function	Shows I/O direction Read (high level) input Write (low level) output
STRB	no function	Strobe pulse, marks valid data

6.6. The parallel interface mechanism (RD/STRB)

The controlling processor (V8) is able to access external components using the GPIO-pins together with the pins RD and STRB as a parallel interface.

In this mode, the GPIO[7:0] are used as 8-bit data bus and the GPIO[11:8] as address bus. The state of the pin RD tells the external device if the V8 is going to read from or write into it. HIGH means “read“, LOW means “write“. In case of a write-access into an external device, this device has to be informed when the data on GPIO are stable (valid). This information provides the STRB-pin. At it’s positive edge, the data are valid and can be read by the external device.

As shown in the timing, it takes 3 clocks (12MHz each) for 1 transaction. The interface, however, can be reprogrammed to reduce the transaction-time to 2 clocks (for example if a fast periphery is attached).

It’s important to note that after finishing an access, the parallel interface releases the I/O-control over the GPIO[11:0]. Now the contents of the I/O-configuration registers V8W_PIO and V8W_PIO2 again take control of the GPIO[11:0]-direction. Therefore it is important to customize these registers according to the external circuitry to avoid driver crashes or floating inputs. A good approach is to program all GPIO’s to output and connect the CS/CE (ChipSelect/ChipEnable) of the external components to the STRB/RD-pins in a way, that their drivers onto GPIO[7:0] are activated only when RD= STRB= HIGH.

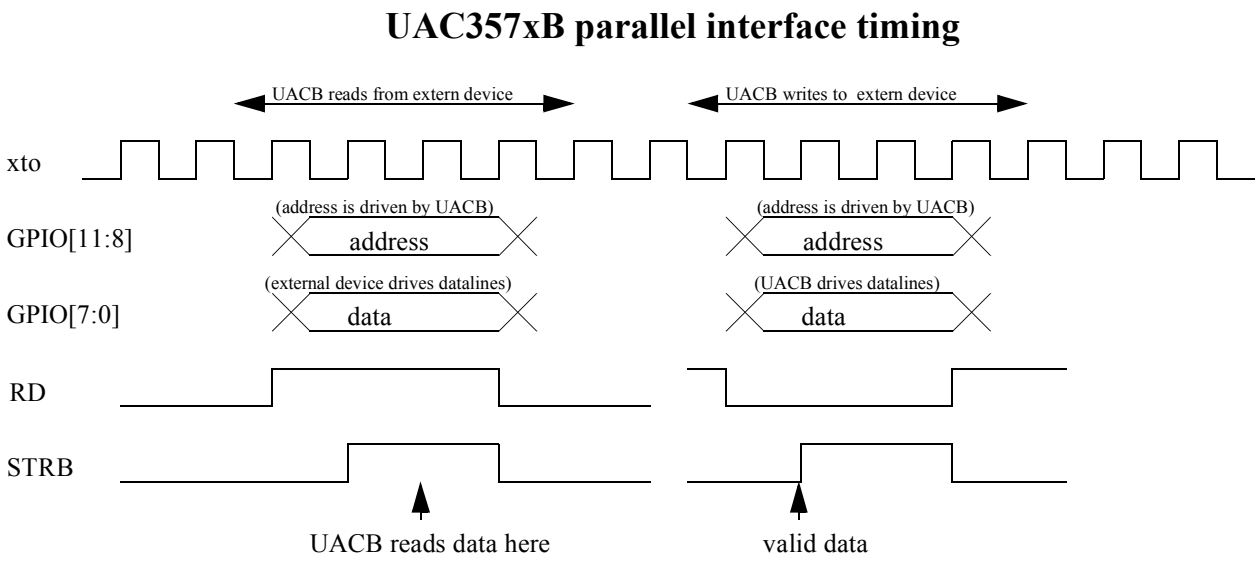

Fig. 6–1: gives an overview of the architecture of the UACB.

7. Alternate Downstream ISO Endpoint

7.1. How to use the alternative downstream ISO-Endpoint

The descriptor

```
.equ    EP2_IFC_NR                              5
        .equ    EP2_OUT_ADR                      0x02
        .equ    USB_EP2_BUFFER_SIZE              0xC0

;;Standard Interface  Descriptor for alternate setting 0
        .byte    0x09                   ;blength          Length of this desriptor
        .byte    4                      ;bDescriptorType  4=INTERFACE
        .byte   EP2_IFC_NR              ;bIfaceNum        interface number
        .byte    0                      ;bAltSetting Alternate setting 0
        .byte    0x00                   ;bNumEndpoints    Number of endpoints= use EP0 only
        .byte    0xFF                   ;bIfaceClass      Interface class
        .byte    0xFF                   ;bIfaceSubClass
        .byte    0xFF                   ;bIfaceProtocol
        .byte    0x00                   ;iInterface       String
;-------------------------------------------------------------------------------

    ;; Standard Interface Descriptor for alternate setting 1
        .byte    0x09                   ; blength         Length of this desriptor
        .byte    4                      ; bDescriptorType 4=INTERFACE
        .byte    EP2_IFC_NR             ; bIfaceNum       interface number
        .byte    1                       ; bAltSetting Alternate setting
        .byte    0x01                   ; bNumEndpoints   Number of endpoints
        .byte    0xFF                   ; bIfaceClass     Interface class
        .byte    0xFF                   ; bIfaceSubClass
        .byte    0xFF                   ; bIfaceProtocol
        .byte    0x00                   ; iInterface      String

   ;;EP2 Out-Endpoint Desriptor
        .byte    7                       ; bLength        Length of this descriptor
        .byte    5                       ; bDescType      ENDPOINT
        .byte   EP2_OUT_ADR             ; bEndpointAddress
        .byte    0x01                    ; bmAttr: its a ISO endpoint
        .rword   USB_EP2_BUFFER_SIZE            ; wMaxPacketSize
        .byte   1                       ; bInterval - for isochronous EPs must be set to 1
```

8. Appendex 1: SDK

-> von W. Platzer bis KW 26

9. Appendix 2: DSP Firmware

9.1. Overview

Figure 9–1 gives an overview of the signal flow inside the DSP firmware. Table 9–1 summarizes the two feature sets.

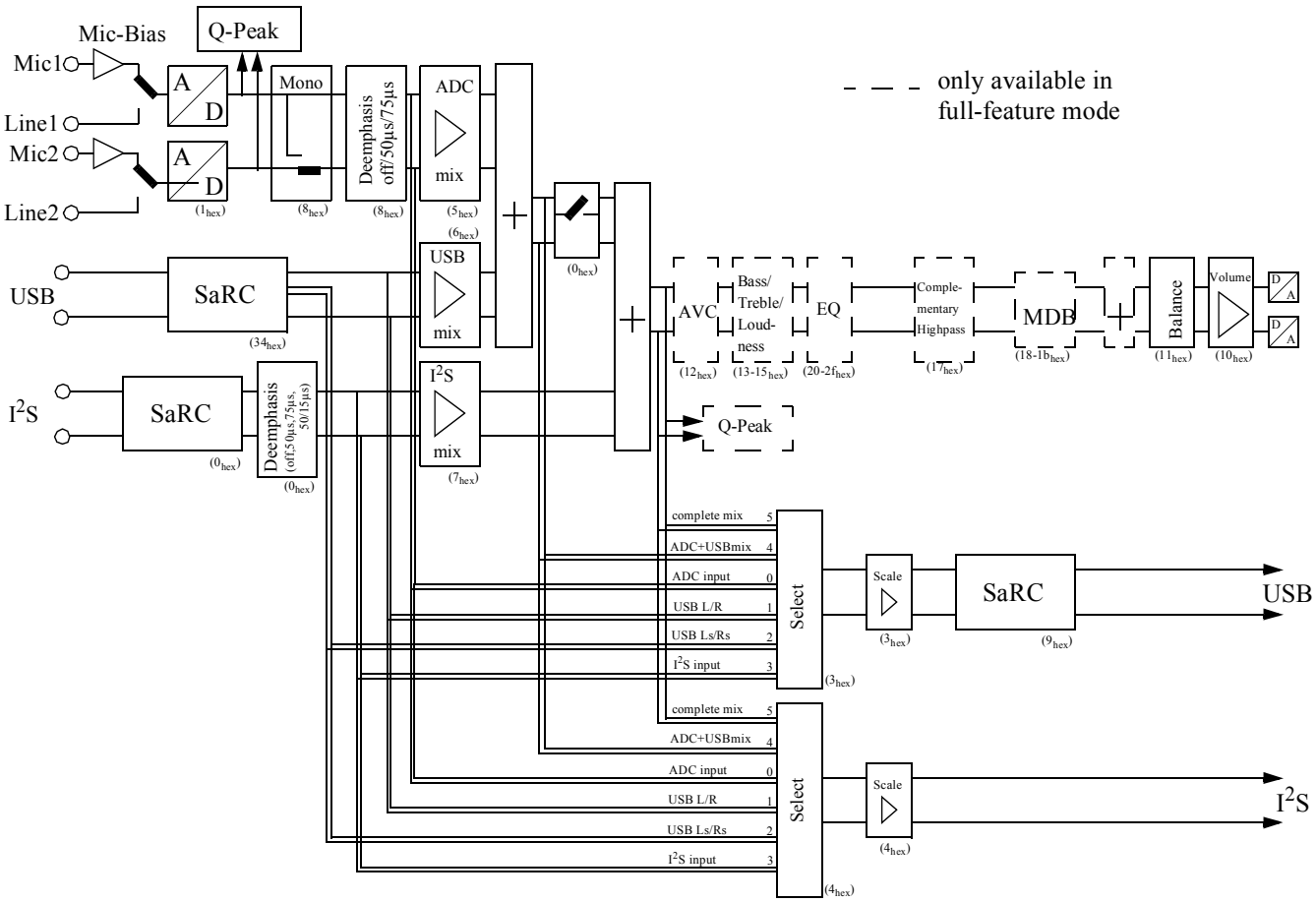

Fig. 9–1: block diagram of XDFP firmware

Table 9–1: Feature set overview ((x) = not implemented in version A0-C2)

features	reduced	full	feature	reduced	full
Mic/Line input incl. Mic-Bias	x	x	Balance	x	x
deemphasis	x	x	Volume	x	x
USB downstream (6.4 .. 48 kHz)	x	x	DAC L/R	x	x
synchr./asyn. I^2S input	x/-	x/(x)			
Mixer	x	x	Micronas Dynamic Bass	-	x
AVC/Bass/Treble/Loudness	-	x	USB upstream (6.4 .. 48 kHz)	x	x
parametric EQ	-	x	synchr. I^2S output	x	x

9.2. User Registers

Table 9–2: DSP Write Registers

Register Address	Function	Name
A/D- D/A CONVERTER CONFIGURATION		
00 01_{hex}	**Converter Configuration (Reset 0x00)** bit [17..14] ADC left amplifier gain bit [13..10] ADC right amplifier gain 1111 +19.5 dB 1110 +18.0 dB 1101 +16.5 dB 1100 +15.0 dB 1011 +13.5 dB 1010 +12.0 dB 1001 +10.5 dB 1000 +9.0 dB 0111 +7.5 dB 0110 +6.0 dB 0101 +4.5 dB 0100 +3.0 dB 0011 +1.5 dB 0010 0.0 dB 0001 -1.5 dB 0000 - 3.0 dB (reset) bit [9] 0/1 Microphone Bias bit [8] 0/1 pseudo Differential Mode Line In right bit [7] 0/1 pseudo Differential Mode Line In left bit [6] 0/1 Line-in R/Microphone R selected for ADC right bit [5] 0/1 Line-in L/Microphone L selected for ADC left bit [4] 0/1 off/on ADC left bit [3] 0/1 off/on ADC right bit [2..1] 0 undefined bit [0] .....0/1..... Headphone off/on (DAC only), (*Default is off).* if headphone dacs are switched on, the headphone opamps must be enabled (V8: Analog Control bit[7]=1).	CONV_CONF

Table 9–2: DSP Write Registers

Register Address	Function	Name
I²S-CONFIG		
00 00_{hex}	**I2S-config (Reset 0x00)** bit [17] co-processor mode 0 : mix of USB, ADC and I2S input to DAC 1 : only I²S input to DAC bit[16] I²S input sync mode 0 : synchronous (f_s=48 kHz) 1 : adaptive slave (f_s=6.4 ... 48 kHz) **Note**: if bit[16] is set, the USB downstream is only supported with decreased performance. bit[15:10] Deemphasis select 0 : deemphasis off 1 : deemphasis 50 µs 2 : deemphasis 75 µs 3 : deemphasis 50/15 µs bit [9:7] reserved, must be set to zero bit [6] I²S input wordstrobe polarity 0 : 1-left, 0-right 1 : 0-right, 1-left bit [5] I²S input wordstrobe strobe justification 0 : WS changes at data word boundaries 1 : WS changes one clock cycle in advance bit [4] I²S output wordstrobe polarity 0 : 1-left, 0-right 1 : 0-right, 1-left bit [3] I²S output wordstrobe strobe justification 0 : WS changes at data word boundaries 1 : WS changes one clock cycle in advance bit [2] I²S output timing generator settings 0 : 2*32 bit (3.072 MHz) 1 : 2*16 bit (1.536 MHz) bit [1:0] reserved, must be set to zero **Note**: bit[9:2] available only in D3-hardware	I2S_CONFIG
MODE SELECT		
00 02_{hex}	**clock and feature set select (Reset 0x01)** bit [17:2] reserved, must be set to 0 bit [1:0] mode 0 full feature set (f_{clk}=72 MHz) 1 reduced feature set (f_{clk}=48 MHz)	DSP_CLK

Table 9–2: DSP Write Registers

Register Address	Function	Name
ANALOG INPUT PROCESSING		
00 08_{hex}	**Input Mode Setting** bit [17] Mono switch 0 stereo through 1 left channel is copied into the right channel bit [15:4] Reserved, must be set to 0 bit [3:2] Deemphasis select 0 deemphasis off 1 deemphasis 50 µs 2 deemphasis 75 µs bit [1:0] Reserved, must be set to 0	ADC_IN_MODE
USB UPSTREAM PARAMETERS		
00 09_{hex}	**USB upstream settings** bit [17:2] Sample Frequency in Hz (e.g. 44.1 kHz = $AC44_{hex}$) bit [1:0] number of channels 00 mono 01 stereo	USB_OUT_FS
USB DOWNSTREAM PARAMETERS		
00 34_{hex}	**USB downstream setting** bit [17:2] Reserved, must be set to 0 bit [1:0] number of channels 0_{hex} mono 1_{hex} stereo 3_{hex} 4 channels (maximum fs=24 kHz !)	USB_IN_CHANS

Table 9–2: DSP Write Registers

Register Address	Function	Name
OUTPUT SOURCE SELECTORS		
00 03_{hex}	**USB upstream source selector** bit [17:10] Scale factor table with 1 dB step size 79_{hex} +6 dB (maximum volume) 78_{hex} +5 dB ... 74_{hex} +1 dB 73_{hex} 0 dB 72_{hex} −1 dB ... 02_{hex} −113 dB 01_{hex} −114 dB 00_{hex} mute (reset condition) bit [9:2] Source Select 0_{hex} ADC input 1_{hex} USB downstream channels 1+2 2_{hex} USB downstream channels 3+4 3_{hex} I^2S input 4_{hex} mix of ADC + USB downstream 5_{hex} complete mix bit [1:0] Reserved, must be set to 0	USB_OUT_SEL
00 04_{hex}	**I^2S source selector** bit [17:10] Scale factor table with 1 dB step size 79_{hex} +6 dB (maximum volume) 78_{hex} +5 dB ... 74_{hex} +1 dB 73_{hex} 0 dB 72_{hex} −1 dB ... 02_{hex} −113 dB 01_{hex} −114 dB 00_{hex} mute (reset condition) bit [9:2] Source Select 0_{hex} ADC input 1_{hex} USB downstream channels 1+2 2_{hex} USB downstream channels 3+4 3_{hex} I^2S input 4_{hex} mix of ADC + USB downstream 5_{hex} complete mix bit [1:0] Reserved, must be set to 0	I2S_OUT_SEL

Table 9–2: DSP Write Registers

Register Address	Function	Name
MIXER UNIT		
00 05_{hex} 00 06_{hex} 00 07_{hex}	**ADC mixing gain** **USB downstream mixing gain** **I^2S mixing gain** bit [17:10] Scale factor table with 1 dB step size 79_{hex} +6 dB (maximum volume) 78_{hex} +5 dB ... 74_{hex} +1 dB 73_{hex} 0 dB 72_{hex} −1 dB ... 02_{hex} −113 dB 01_{hex} −114 dB 00_{hex} mute (reset condition) bit [9:0] Reserved, must be set to 0	MIX_ADC MIX_USB MIX_I2S
BASEBAND PROCESSING		
00 13_{hex}	**Bass** bit [17:10] Bass range 60_{hex} +12 dB 58_{hex} +11 dB ... 08_{hex} +1 dB 00_{hex} 0 dB $f8_{hex}$ −1 dB ... $a8_{hex}$ −11 dB $a0_{hex}$ −12 dB Higher resolution is possible, one LSB step results in a gain step of about 1/8 dB. With positive bass settings clipping of the output signal may occur. Therefore, it is not recommended to set bass to a value that results in conjunction with volume to an overall positive gain. bit [9:0] Reserved, must be set to 0	BASS

Table 9–2: DSP Write Registers

Register Address	Function	Name
00 14_{hex}	**Treble** bit [17:10] Treble range 60_{hex} +12 dB 58_{hex} +11 dB ... 08_{hex} +1 dB 00_{hex} 0 dB $f8_{hex}$ −1 dB ... $a8_{hex}$ −11 dB $a0_{hex}$ −12 dB Higher resolution is possible, one LSB step results in a gain step of about 1/8 dB. With positive treble settings, clipping of the output signal may occur. Therefore, it is not recommended to set treble to a value that results in conjunction with volume to an overall positive gain. bit [9:0] Reserved, must be set to 0	TREBLE
00 15_{hex}	**Loudness** bit [17:10] Loudness Gain 44_{hex} +17 dB 40_{hex} +16 dB ... 04_{hex} +1 dB 00_{hex} 0 dB bit [9:2] Loudness Mode 00_{hex} normal (constant volume at 1 kHz) 04_{hex} Super Bass (constant volume at 2 kHz) bit [1:0] Reserved, must be set to 0 Higher resolution of Loudness Gain is possible: An LSB step results in a gain step of about 1/4 dB. Loudness increases the volume of low- and high-frequency signals, while keeping the amplitude of the 1-kHz reference frequency constant. The intended loudness has to be set according to the actual volume setting. Because loudness introduces gain, it is not recommended to set loudness to a value that, in conjunction with volume, would result in an overall positive gain. The corner frequency for bass amplification can be set to two 2 different values. In normal mode, the point of constant volume is at 1kHz. In Super Bass mode, the point of constant volume is at 2kHz.	LDNESS

Table 9–2: DSP Write Registers

Register Address	Function	Name
00 $0e_{hex}$	**D/A output mode** bit [17] Mono switch 0 stereo through 1 mono matrix applied bit [16] Invert right channel 0 through 1 right channel is inverted bit [15:0] Reserved, must be set to 0 In order to achieve more output power a single loudspeaker can be connected as a bridge between pins OUTL and OUTR. In this mode bit[17] and bit[16] must be set.	DAC_OUT_MODE

Table 9–2: DSP Write Registers

Register Address	Function	Name
MICRONAS DYNAMIC BASS (MDB)		
00 18_{hex}	**MDB Effect Strength** bit[17:10] 00hex MDB OFF (default) 7Fhex maximum MDB The MDB effect strength can be adjusted in 1dB steps. A value of 44hex will yield a medium MDB effect. bit[9:0] 00hex must be zero	MDB_STR
00 19_{hex}	**MDB Harmonic Content** bit[17:10] 00_{hex} no harmonics are added (default) 40_{hex} 50% fundamentals + 50% harmonics $7F_{hex}$ 100% harmonics The MDB exploits the psychoacoustic phenomenon of the 'missing fundamental' by creating harmonics of the frequencies below the highpass frequency of the MDB (MDB_HP). This enables a loudspeaker to reproduce frequencies that are below its cutoff frequency. The Variable MDB_HMC describes the ratio of the harmonics towards the original signal. bit [9:0] 00_{hex} must be zero	MDB_HMC
00 $1a_{hex}$	**MDB Center Frequency** bit[17:10] 2 20 Hz 3 30 Hz ... 30 300 Hz The MDB center frequency defines the center frequency of the MDB bandpass filter. bit[9:0] 00_{hex} must be zero	MDB_FC
00 $1b_{hex}$	**MDB Amplitude Limit** bit[17:10] 00_{hex} 0 dbFS (default, no limitation) FF_{hex} –1 dbFS ... $E0_{hex}$ –32 dbFS The MDB Amplitude Limit defines the maximum allowed amplitude at the output of the MDB relative to 0 dbFS. If the amplitude exceeds MDB_LIM, the gain of the MDB is automatically reduced. Set this value to avoid overloading the speakers. bit[9:0] 00_{hex} must be zero	MDB_LIM

Table 9–2: DSP Write Registers

Register Address	Function	Name
VOLUME		
00 10_{hex}	**Main Volume Control** bit [17:10] Volume table with 1 dB step size $7f_{hex}$ +12 dB (maximum volume) $7e_{hex}$ +11 dB ... 74_{hex} +1 dB 73_{hex} 0 dB 72_{hex} −1 dB ... 02_{hex} −113 dB 01_{hex} −114 dB 00_{hex} mute (reset condition) bit [9:0] Reserved, must be set to 0 This main volume control is applied to the analog outputs only. It is split between a digital and an analog function. In order to avoid noise due to large changes of the setting, the actual setting is internally low-pass filtered. With large scale input signals, positive volume settings may lead to signal clipping.	VOLUME
00 11_{hex}	**Balance** bit [17:10] Balance range $7F_{hex}$ Left −127 dB, Right 0 dB $7E_{hex}$ Left −126 dB, Right 0 dB ... 01_{hex} Left −1 dB, Right 0 dB 00_{hex} Left 0 dB, Right 0 dB FF_{hex} Left 0 dB, Right −1 dB ... 81_{hex} Left 0 dB, Right −127 dB 80_{hex} Left 0 dB, Right −128 dB bit [9:0] Reserved, must be set to 0 Positive balance settings reduce the left channel without affecting the right channel; negative settings reduce the right channel leaving the left channel unaffected.	BALANCE
00 12_{hex}	**Automatic Volume Correction (AVC) Loudspeaker Channel** bit [17:14] 00_{hex} AVC off (and reset internal variables) 08_{hex} AVC on bit [13:10] 08_{hex} 8 sec decay time 04_{hex} 4 sec decay time 02_{hex} 2 sec decay time 01_{hex} 20 ms decay time (intended for quick adaptation to the average volume level after track or source change) bit [9:0] Reserved, must be set to 0 **Note:** To reset the internal variables, the AVC should be switched off and then on again during any track or source change. For standard applications, the recommended decay time is 4 sec.	AVC

Table 9–3: DSP Read Registers

Register Address	Function	Name
ADC QUASI-PEAK DETECTOR READOUT		
00 $0a_{hex}$	**A/D Converter Quasi-Peak Detector Readout Left** bit [14:0] positive 15-bit value, linear scale	QPEAK_L
00 $0b_{hex}$	**A/D Converter Quasi-Peak Detector Readout Right** bit [14:0] positive 15-bit value, linear scale	QPEAK_R
DAC QUASI-PEAK DETECTOR READOUT		
00 $0c_{hex}$	**Audio Processing Input Quasi-Peak Detector Readout Left** bit[14..0] positive 15-bit value, linear scale	DQPEAK_L
00 $0d_{hex}$	**Audio Processing Input Quasi-Peak Detector Readout Right** bit[14..0] positive 15-bit value, linear scale	DQPEAK_R

10. Appendix 3: V8 Controller Registers

V. Summa, T. Ruhnau, D. Bächer, W. Platzer, C. Noeske, M. Winterer

UAC357xB - V8 reference

10.1.Overview

Figure 10–1 gives an overview of the architecture of the UACB.

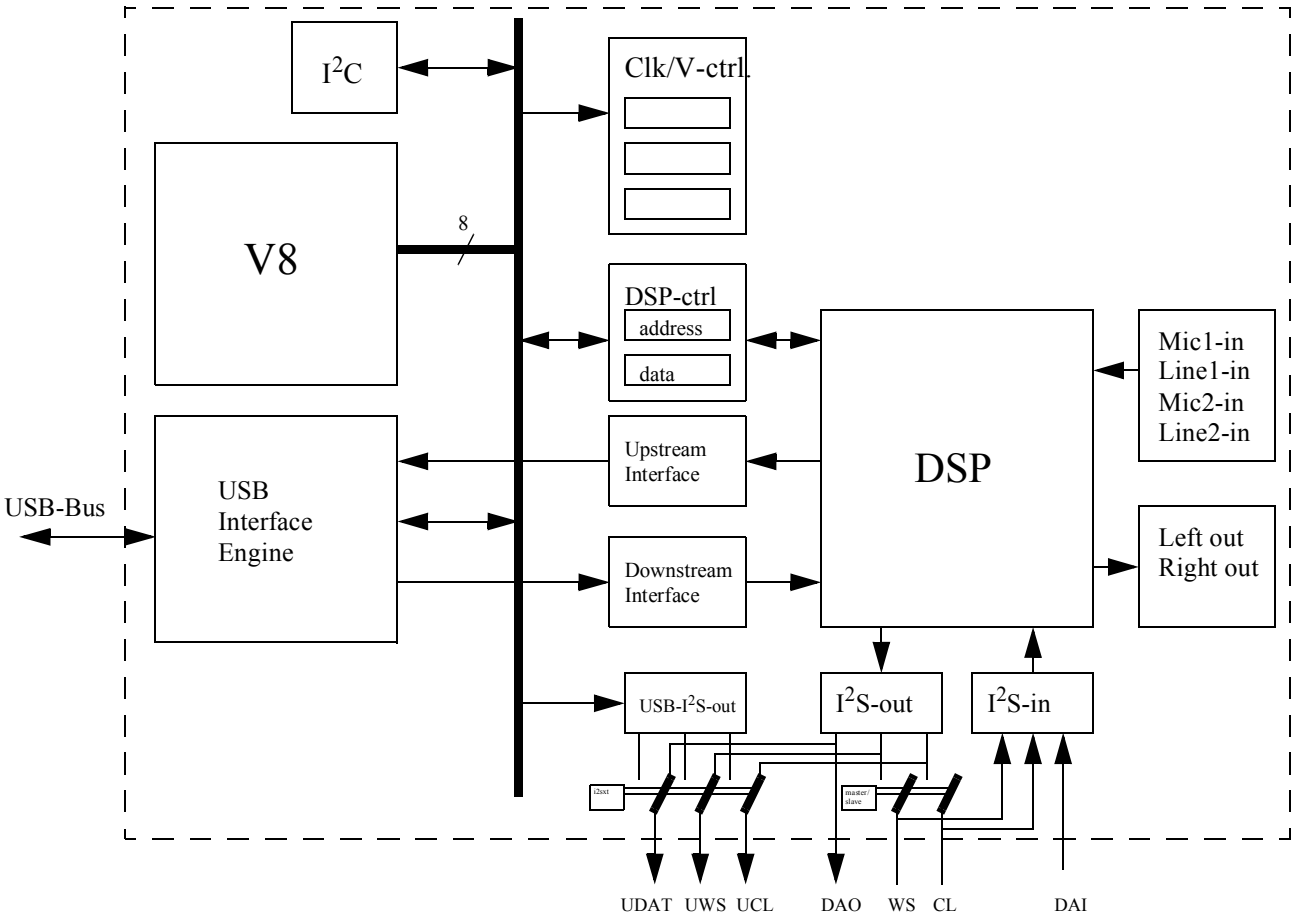

Fig. 10–1: UACB architecture

10.2.V8 Memory Layout

Table 10–1:

Adress	Function
$0000_{hex...}2fff_{hex}$	**12 kbytes ROM,** lowest 8 kbytes are covered by V8 EMU RAM, area above 8k is used by boot loader and some other boot time routines
$8000_{hex...}9fff_{hex}$	**8 kbytes EMU RAM** (mapped to lowest ROM area if v8_emu_reg[0] == 1
$a000_{hex...}a7ff_{hex}$	**2 kbytes RAM** $a000_{hex}...a07f_{hex}$:128 bytes BDT (16 bytes per EP) $a080_{hex}...a0ff_{hex}$:128 bytes jump table $a100_{hex}...a10f_{hex}$:16 bytes interrupt vectors $a110_{hex}...a17f_{hex}$:112 bytes stack $a180_{hex}...a200_{hex}$:128 bytes variables EP7 is not being implemented so the respective 16 bytes are used for variable space. The lowest 32 bytes of the stack is also used for variables If the boot loader finds a plugin within an EEPROM the first data byte of the plugin is interpreted as offset within the jump table. The respective ‘jmp’ command is substituted with a jump to the plugin space. The RAM-clear routine clears from $a000_{hex}...a23f_{hex}$

10.3.V8 Peripherals & Control & Status Registers

10.3.1. V8 General & Interface Registers

Table 10–2: V8 General & Interface Registers

Register Address	Function	Name
$b000_{hex}$	**USB control register** (rw) bit[7:4] do not use (the highest prioritized interrupts (ro)) bit[3] = 1: Vbus (ro) bit[2] = 1: suspend marker bit[1] = 1: USB activity detected, set by non-idle state bit[0] = 1: suspend mode, reset by non-idle state via combinational logic	usb_ctl_reg
$b001_{hex}$	**V8 control register** (rw) bit[7] = 1: GPIO[[11] is pwm_out, = 0: GPIO[11] has normal function bit[6] = 1: flag used in V8 software: dont run any schedulers bit[5] = 1: flag used in V8 software: dont run the boot loader bit[4] = 1: use alt_iso_out, = 0 : use 'dumb_serial_interface' i.e. I2S like bit[3] = 1: pullup D+ (USB connected), = 0 : D+ floating (USB disconnected) bit[2] : access to sie 0: two wait states 1: three wait state bit[1] : access to gpb 0: two wait states 1: one wait state bit[0] : access to i2c 0: two wait states 1: one wait state	v8_ctl_reg
$b002_{hex}$	**V8 interrupt enable register** (rw) **bit assignment for default priority scheme (0)** bit[7:5] priority code (8 schemes) bit[4] = 1: enable i2c_int, lowest prio, maskable by PSR[3] bit[3] = 1: enable usb_int, maskable by PSR[3] bit[2] = 1: enable timer_int, maskable by PSR[3] bit[1] = 1: enable im_int (from XDFP) 2nd highest prio, maskable by PSR[3] bit[0] = 1: enable test_int, nonmaskable with highest priority	v8_int_enb

Table 10–2: V8 General & Interface Registers

Register Address	Function	Name
$b003_{hex}$	**V8 emu register** (rw) bit[7]: flag used in V8 SW: I2C slave maps subaddr ’0000’ to ’4000’ -> /dev/null bit[6:4] : boot error codes - visible at {usb_sws, usb_sdo, usb_sck} 000: boot loader terminated without error 001: no external EEPROM found after 5 retries 010: no external nor internal EEPROM found, no header loaded 011: header loaded but then failed loading section_1 100: header loaded but then failed loading section_2 101: header loaded but then failed loading section_3 110: header loaded but then failed loading section_4 111: persisting SCL_down or skipping the boot loader altogether bit[3] = 1: (reserved) bit[2] = 1: hard reset I2C only, reset after 16 clock cycles bit[1] = 1: hard reset V8 only, reset after 16 clock cycles bit[0] = 1: swap decoding of rom_ce and emu_ram_ce	v8_emu_reg
$b004_{hex}$	**V8 hardware config register** (rw) bit[7] = 1: enable timer by sofo_ll bit[6] = 1: enable timer by usb_iso_reado bit[5] = 1: enable timer by usb_iso_writeo bit4] = 1: enable timer by alt_usb_int (if enabled) bit[3] = 0: test_bus_gnt waits 1 cycle after VUSB ISO access 1: test_bus_gnt immediately after VUSB ISO access bit[2] = 1: one_clk (makes clk_sie == clk_cpu) bit[1] = 1: v8_disable (grants the bus unconditionally to the test interface) bit[0] = 1: dont generate pstrobe pulses for all HW-addresses (b000...b0ff)	v8_hw_config

Table 10–2: V8 General & Interface Registers

Register Address	Function	Name
16 Bit Timer		
$b008_{hex}$	**Timer counter register low** (rw)	timer_cnt_low
$b009_{hex}$	**Timer counter register high** (rw)	timer_cnt_high
$b00a_{hex}$	**Timer CTRL register** (rw) The current count value in the CTR can be read or written at anytime. Care must be taken when accessing the CTR when RUN is 1. No protection is provided to insure that the counter won't count in the time between reading the low and the high halves of the CTR. Generally, you should read the CTRH, then the CTRL, then reread the CTRH and compare with the first read of the CTRH. If the values are equal, then value read is correct. If the values differ, you may want to simply reread the CTR until they do match, or inspect CTRL and select the first CTRH if CTRL[7]=1 or the second CTRH if CTRL[7]=0. bit[7] r = 1: timer interrupt (max reached), w = 1: reset irq bit bit[6] = 1: enable timer interrupt, = 0: mask interrupt bit[5] = 1: counting to MAXB, = 0: counting to MAXA bit[4] = 1: retrigger when cnt=max reached, = 0: stop when cnt=max bit[3] = 1: enable MAXB, = 0: use only MAXA bit[2] (reserved), set to zero bit[1] = 1: use EXT_ENB to enable counter, = 0: internal clock bit[0] = 1: enable counter, = 0: stop counter (RUN)	timer_ctl
$b00b_{hex}$	**Timer prescaler register** (rw) Divide the selected count enable by the value in this register plus one. 0=disable prescale, 1=divide by 2, 2=divide by 3, 255=divide by 256.	timer_presc_reg
$b00c_{hex}$	**Timer MAXA low register** (rw)	timer_maxa_low
$b00d_{hex}$	**Timer MAXA high register** (rw)	timer_maxa_hi
$b00e_{hex}$	**Timer MAXB low register** (rw)	timer_maxb_low
$b00f_{hex}$	**Timer MAXB high register** (rw)	timer_maxb_hi
I2C Master/Slave Interface		
$b010_{hex}$	**I2C control register** (rw) mostly used for setting operation modes and indicating protocol phases bit[7] = 1: end_of_telegram (marks the last byte of the telgram) bit[6] (reserved) bit[5] = 1: repeat_start (generate a repeated START condition) bit[4] = 1: ignore_ack (never checks acknowledge, just reports bus_errors when no ack was seen) bit[3] = 1: enable I2C interrupt bit[2] = 1: master_enable, set before starting a master telegram, can be released after writing the I2C address bit[1] = 1: fast mode, = 0: slow mode bit[0] = 1: enable power down mode, I2C clock will stop if no transfer is active	i2c_ctl_reg

Table 10–2: V8 General & Interface Registers

Register Address	Function	Name
$b011_{hex}$	**I2C data register** (rw) writing a byte to the Data register turns the I2C interface into master mode and issues a bus_request if the bus is currently free.	i2c_data_reg
$b012_{hex}$	**I2C interrupt status register** interrupt status and device status indication bit[7] r = 1: loss of arbitration or no response from any device (see bus_active), w = 1: reset irq bit bit[6] r =1: we transmitted a read device_ID (maybe we must set "end_of_telegram" now), w = 1: reset irq bit bit[5] r = 1: buffer_full (has received a new data byte), w = 1: reset irq bit bit[4] r = 1: buffer_empty (needs a new data byte to be sent), w = 1: reset irq bit bit[3] r = 1: bus_error (I2C bus hangs and is unusable), w = 1: reset irq bit bit[2] (ro) = 1: first data byte received or transmitted bit[1] (ro) = 1: Interface is active - slave or master bit[0] (ro) = 1: Interface is in master mode	i2c_int_stat_reg
$b013_{hex}$	**I2C device_ID register (left aligned)** (rw) writing a byte to this register sets the device ID of the I2C interface, LSB is ignored	i2c_devid_reg
USB Serial Data Output Interface		
$b018_{hex}$	**USB DAT Serial Interface** (rw) bit[7:0] Data to be written. serial out MSB first Interface Control: {polarity, rise_cnt[2:0], fall_cnt[3:0]}	dsi_reg
$b019_{hex}$	**USB DAT Serial Interface** (rw) bit[7:0] read: {empty, 3'b0, bit_cnt[3:0]} write: {polarity, rise_cnt[2:0], fall_cnt[3:0]}	
Audio Streaming Interface		
$b030_{hex}$	**Audio Streaming Interface, test low byte** read next upstream byte - low write next downstream byte - low (first on USB bus)	audio_test_low
$b031_{hex}$	**Audio Streaming Interface, test middle byte** read next upstream byte - mid write next downstream byte - mid	audio_test_mid
$b032_{hex}$	**Audio Streaming Interface, test high byte** read next upstream byte - high write next downstream byte - high, shift into fifo	audio_test_high
$b033_{hex}$	**Audio Streaming Interface, all bytes** this is the data source/sink for the two ISO interfaces, controlled by xfifo_ctl[3:0]	audio_test_all

Table 10–2: V8 General & Interface Registers

Register Address	Function	Name
$b034_{hex}$	**Audio downstream format** bit[2] = 1: 4 channel audio (regardless of bit[1]) bit[1] = 1: stereo, = 0: mono (24 bit mono not supported) bit[0] = 1: 24 bit resolution, = 0: 16 bit resolution	audio_dwn_fmt
$b035_{hex}$	**Audio upstream format** bit[1] = 1: stereo, = 0: mono (8 bit stereo not supported) bit[0] = 1: 16bit resolution, = 0: 8bit resolution interface is locked if bit[1:0] = '10'	audio_up_fmt
$b037_{hex}$	**Audio Interface Status** (ro) bit[7] 0 bit[6] downstream full bit[5] upstream empty bit[4:0] reserved	audio_up_fmt
XDFP-CONTROL-INTERFACE		
$b042_{hex}$	**High-part of XDFP-data** (wo) bit[7:0] DSP reads these bits as bit[17:10] from I2C_DATA_IN	V8W_DHI
$b043_{hex}$	**Mid-part of XDFP-data** (wo) bit[7:0] DSP reads these bits as bit[9:2] from I2C_DATA_IN	V8W_DMID
$b044_{hex}$	**Low-part of XDFP-data** (wo) bit[7:2] reserved, should be set to zero bit[1:0] DSP reads these bits as bit[1:0] from I2C_DATA_IN	V8W_DLO
$b045_{hex}$	**High-part of XDFP-address & command-control** (wo) bit[5] DSP direct register access 0 : memory 1 : register bit[3:2] DSP command 00 : write 01 : read 10 : Jump to address in I2C_DATA_IN 11 : reserved, must not be used bit[1:0] DSP interprets these bits as bit[9:8] of the address	V8W_AHI
$b046_{hex}$	**Low-part of XDFP-address** (wo) bit[7:0] DSP interprets these bits as bit[7:0] of the address	V8W_ALO
$b05c_{hex}$	**XDFP data high (Reset: 0_{hex})** bit[7:0] data: contains bits[19:10] of the XDFP I2C_DATA_OUT register	V8R_DHI
$b05d_{hex}$	**XDFP data high (Reset: 0_{hex})** bit[7:0] data: contains bits[8:2] of the XDFP I2C_DATA_OUT register	V8R_DMID
$b05e_{hex}$	**XDFP data high (Reset: 0_{hex})** bit[7:0] data: contains bits[1:0] of the XDFP I2C_DATA_OUT register	V8R_DLO

Table 10–2: V8 General & Interface Registers

Register Address	Function	Name
MISC REGISTER		
$b05f_{hex}$	**Power-Modes / DSP Status Register** (ro) Reset: $x0000001_{bin}$, depending on SEN-Pin bit[7] suspend enable : state of the “SEN”-pin bit[6] suspend marker: this bit delivers the state of V8W_APP[6] 0 : normal operation 1 : IC is about to enter,is already in or has just left a suspend mode triggered by the USB-host. bit[5] remote wakeup marker: 0 : normal operation 1 : suspend triggered by "external wake-up" bit[4] usb_sdo: used for selection of the EEPROM dev ID bit[3] usb_sck: used for selection of fail-safe mode bit[2] read active: DSP read, XDFP-control-interface-flag 0 : inactive 1 : active bit[1] write active: DSP write, XDFP-control-interface-flag 0 : inactive 1 : active bit[0] busy : DSP control interface is busy 0 : inactive 1 : active	V8R_STAT
$b06f_{hex}$	**UACB i2c-device-address initialization** bit[7:3] reserved, must be set to zero bit[2] simple_i2c; pin USBWSO bit[1] failsave; pin USBDAT bit[0] i2caddr; pin USBCLK	V8R_ICA

10.4. USB Serial Engine Interface

(see VUSB Documentation)

Table 10–3: USB Serial Engine Interface

Register Address	Function	Name
$b080_{hex}$	**SIE interrupt status** (rw) read = {2’b0, resume, sleep, tok_dne, sof_tok, error, usb_rst} writing a ‘1’ resets the respective bit, writing to this location pops one byte out of hte usb_stat fifo	usb_int_stat
$b081_{hex}$	**SIE interrupt enable** (rw) {2’b0, resume, sleep, tok_dne, sof_tok, error, usb_rst}	usb_int_enb

Table 10–3: USB Serial Engine Interface

Register Address	Function	Name
$b082_{hex}$	**SIE error status** (rw) {bts, own, dma, bto, dfn8, crc16, crc5, pid} writing a '1' resets the respective bit	usb_err_stat
$b083_{hex}$	**SIE error enable** (rw) {bts, own, dma, bto, dfn8, crc16, crc5, pid}	usb_err_enb
$b084_{hex}$	**SIE status** (ro) {endp[3:0], in, odd, 0, 0}	usb_stat
$b085_{hex}$	**SIE control** (rw) {5'b0, resume, odd_rst, usb_en}	usb_ctl
$b086_{hex}$	**SIE address** (rw) {1'b0, addr[6:0]}	usb_addr
$b087_{hex}$	**SIE BDT page** (rw) for UACB this is a constant 'a0'	usb_bdt_page
$b08c_{hex}$	**SIE Micronas control register** (rw) bit7:4] current token (ro) bit[3:2] setting for usb_pre_bus_lock duration (rw) 00 : early buslock only for ISO IN tokens 01 : very early buslock only for ISO IN tokens (2 cycles earlier than "early") 10 : early buslock for every IN and OUT tokens 11 : very early buslock for all ISO IN and ISO OUT tokens, early buslock for non-ISO IN and OUT tokens bit[1] = 1: enable new sync detection circuitry built into DPLL bit[0] = 1: enable SOF fly wheel	usb_sie_control
$b08d_{hex}$	**SIE micronas status register** (ro) bit[4] current endpoint bit[3] isochronous endpoint bit[2] own bit[1] endpoint rx odd bit[0] endpoint tx odd	usb_sie_status
$b08e_{hex}$	**SIE micronas configuration register** (rw) bit7:4] not defined, set to zero bit[3] = 1: don't stall after 'BDT status' - default is 'stall all non-SETUP after status' bit[2] = 1: stall 'DATA0 status' - default is dont stall bit[1] = 1: disable 'bdt_stall' - default is enable bit[0] = 1: 'crc16_err_set' doesn't break the rx transfer	usb_sie_config

Table 10–3: USB Serial Engine Interface

Register Address	Function	Name
$b090_{hex}$	**SIE enpoint 0 control register** (rw) bit[7:6] reserved bit[5] r = 1: new SETUP for even BDT, w = 1: reset this bit bit[4] r = 1: new SETUP for odd BDT, w = 1: reset this bit bit[3] = 1: enable OUT transactions for this endpoint bit[2] = 1: enable IN transactions for this endpoint bit[1] = 1: stall this endpoint bit[0] = 1: enable handshake for this endpoint	usb_ep0_ctl
$b09x_{hex}$	**SIE enpoint x (x = 1..7) control register** (rw) bit[7] = 1: disable buffer toggling for this endpoint bit[6:5] reserved bit[5] = 1: isochronous EP bit[3] = 1: enable OUT transactions for this endpoint bit[2] = 1: enable IN transactions for this endpoint bit[1] = 1: stall this endpoint bit[0] = 1: enable handshake for this endpoint	usb_epx_ctl

10.5. General Purpose IO Registers / Test Bus IF

Table 10–4: General Purpose IO Registers / Test Bus IF

Register Address	Function	Name
$b0a0_{hex}$	**test data / GPO[7:0]** (rw) read/write to the output register, to see data at the pins the output direction must be set.	gpo_low
$b0a1_{hex}$	**test addr_low / { GPO[11:8] , GPI[11:8] }** (rw) read from the GPIO pins / write to the output register, to see data at the pins the output direction must be set.	gpio_hi
$b0a2_{hex}$	**test addr_high / GPI[7:0]** (ro) read the data at the GPIO pins	gpi_low
$b0a3_{hex}$	**test interrupt done** (wo) writing to this location releases trdy, thus indicating that the test interrupt has been completely serviced by the V8.	test_int_dne
$b0b0_{hex}$ . . $b0bf_{hex}$	**GPIO parallel bus interface, access to GPIO address / data lines** writes/reads directly to/from GPIO {paddr[3:0], pdata[7:0]}, see 'gpb_strobe' and 'gpb_rwq' (uses the same resources as the test interface so, again: be careful not to interfere with testmode)	

10.6. Application specific registers

The following Table shows write only hardware register of the UAC357xB. Register shadows are kept in the V8 RAM and are cyclically written to the hardware register by the processor.

Table 10–5: V8 Write Only Registers

Register Address	Function	Name
SOFTRESETS		
$b04e_{hex}$	**Software-Reset (Reset: 0)** bit[7:2] reserved, should be set to zero bit[1] dfpres : DSP reset 0 : normal mode 1 : permanent reset bit[0] por : Processor reset 0 : normal mode 1 : reset of DSP and V8	V8W_SOFTRES
MAIN CONTROL		
$b04f_{hex}$	**Application (Reset: 10_{hex})** Shadow: **V8_APP_REG_SHADOW**= $a180_{hex}$ This register is not updated by the processor bit[7] i2sxt :source select of USB streaming interface 0 : V8 uses USB-I^2S-Interface (usbwso/usbclk) 1 : DSP I^2S-output-interface (usbwso/usbclk) bit[6] spden :SusPenDENable, 0 : normal state 1 : enables the suspend mode logic **Note:** If this bit is set, the suspend mode is entered when the synthesis-part activates its “suspd“-wire The bit can be reread as bit[6] of the status register V8RSTAT. The bit is not resetted during suspend. bit[5] resxwu :reset external-wakup-flag (=bit[5] of V8W_STAT) 0 : normal state 1 : reset **Note:** A LOW-state at the UACB-pin SEN sets an internal flip-flop, meaning an “external wakup request“.This flip-flop can be resetted by setting the “resxwu“-bit. The bit is not resetted during suspend. bit[4:3] xdfreq : set XDFP frequency (version 0201 and higher) 00 : 72MHz 01 : 48MHz 10 : 36MHz (default after reset) 11 : 24MHz bit[2] sofbyp : SOF bypass 0 : SOF from synthesis 1 : SOF comes from SOF-Pad bit[1:0] mclksel : freq of pin "mclk" 00 : 18 MHz 01 : 24 MHz 10 : 36 MHz 11 : 48 kHz-IRQ (debug & test)	V8W_APP

Table 10–5: V8 Write Only Registers

Register Address	Function	Name
$b05b_{hex}$	**Analog Control (Reset: $c0_{hex}$)** Shadow: **V8_ANCTR_SHADOW**= $a181_{hex}$ bit[7] outlron : 0 : disable headphone opamp 1 : enable headphone opamp, force sref to on state (default) bit[6] srefon : 0 : sref off 1 : sref on (default) bit[5] filton : 0 : disable filter opamp 1 : enable filter opamp bit[4] reserved, must be set to zero bit[3] Internal reset enable bit[2] Pseudo differential output mode bit[1] Common output mode bit[0] setagn : set voltage of sref 0 : 1.725V 1 : 2.3V	V8W_ANCTR
I/O-PIN CONTROL		
$b050_{hex}$	**I/O-Control (Reset: 0)** Shadow: **V8W_IO_SHADOW=** $a18a_{hex}$ 0 : input/tristate 1 : output bit[7] strbdrv :pin "strb" bit[6] rddrv :pin "rd" bit[5] usbddrv :pin "usbdat" bit[4] usbcldrv :pin "usbclk" bit[3] vbusdrv :pin "vbus" bit[2] clidrv :pin "cli" bit[1] wsidrv :pin "wsi" bit[0] daidrv :pin "dai"	V8W_IO

Table 10–5: V8 Write Only Registers

Register Address	Function	Name
$b057_{hex}$	**I/O-Control2 (Reset: $0f_{hex}$)** Shadow**: V8W_IO2_SHADOW=** $a18b_{hex}$ 0: input/tristate 1: output bit[7] mclkdrv :pin "mclk" bit[6] daodrv :pin "dao" bit[5] usbwsdrv :pin "usbws" bit[4] sendrv :pin "sen" bit[3] vbuspd: enables PULLDOWN of pin VBUS 0 : pulldown disabled 1 : pulldown enabled (default) bit[2] trdydrv :pin "trdy" bit[1] sofdrv :pin "sof" bit[0] suspdrv :pin "suspend" **Note:** this register is only reset by the RESQ-pin (NOT during suspend mode)	V8W_IO2
$b058_{hex}$	**GPIO-Control (Reset: 0)** Shadow**: GPIO_CONFIG_HI_SHADOW=** $a185_{hex}$ bit[7:4] not used bit[4:0] gpiohidrv : configure direction of gpio[11:8]-pins 0 : input / tristate 1 : output	V8W_PIO2
$b051_{hex}$	**GPIO-Control (Reset: 0)** Shadow**: GPIO_CONFIG_LO_SHADOW=** $a184_{hex}$ bit[7:0] gpiodrv : configure direction of gpio[7:0]-pins 0 : input / tristate 1 : output	V8W_PIO
$b055_{hex}$	**GPIO[7:0]-Driver strength (Reset: 0)** Shadow**: GPIO_PADSTRENGTH_LO_SHADOW=** $a188_{hex}$ bit[7:0] gpio : set pad driver strength of gpio[7:0]-pins 0 : weak 1 : strong	V8W_PS
$b056_{hex}$	**DriverStrength (Reset: 0)** Shadow**: GPIO_PADSTRENGTH_HI_SHADOW=** $a189_{hex}$ pad-strength of gpio[11:8] and other pins 0 : weak 1 : strong bit[7] i2ss : driver strengths of the pins dao,dai,wsi,cli bit[6] usbs : driver strengths of the pins usbwso,usbclk,usbdat bit[5] sofs : driver strengths of the pins sof,sen,suspendq,trdyI bit[4] strbs : driver strengths of the pins strb,rd bit[3:0] gpiohi : driver strengths of the gpio[11:8]	V8W_PS2

Table 10–5: V8 Write Only Registers

Register Address	Function	Name
$b059_{hex}$	**Pulldown-Control (Reset: ff_{hex})** Shadow**: GPIO_PULLDOWN_LO_SHADOW=** $a186_{hex}$ bit[7:0] gpiolo : configure pulldowns of gpio[7:0]-pins 0 : no pulldown 1 : pulldown	V8W_PDEN
$b05a_{hex}$	**Pulldown-Control of GPIO[11:8] and other pins (Reset: ff_{hex})** Shadow**: GPIO_PULLDOWN_HI_SHADOW=** $a187_{he}$ bit[7] mclk : enable/disable pulldown of pin mclk bit[6] dao : enable/disable pulldown of pin dao bit[5] usbif : enable/disable pulldown of usb serial interface (pins usbdat, usbws and usbclk) bit[4] pif : enable/disable pulldown of parallel interface (pins strb and rd) bit[3:0] gpiohi : enable/disable pulldown of gpio[11:8] Note: this register is only resetted by RESQ-pin (NOT during suspend!)	V8W_PDEN2
$b052_{hex}$	**Analog volume delay adjust (Reset: 35dez)** This register sets the time, after which the analog DAC-volume values, which are written by the XDFP, are applied to the DAC volume circuitry. bit[7:0] avdly : analog-delay-adjustion, reset = 35 (2.9us) **Unit:** 12MHz-steps = 83.3 ns **Example:** a value of avdly=20 delays the volume by 20*83.3ns= 1666ns= 1.6us	V8W_AVDLY

Table 10–5: V8 Write Only Registers

Register Address	Function	Name
$b078_{hex}$	**Voltage regulators (Reset: 00_{hex})** Properties of digital voltage regulator(s) bit[7] bv5v : analog-PA 0 : 3.5V 1 : voltage regulator bridged (analog at 5V=extern voltage) bit[6] unused (0101- 0201: av5v : analog-part 0..3.5V 1..bridged) bit[5] v85v : V8 voltage regulator 0 : 3.75V 1 : v8-voltage regulator bridged (v8 at 5V=extern voltage) bit[4] v8poff : v8-voltage regulator 0 : enable 1 : disable bit[3] x5v : DSP voltage regulator 0 : 3.75V 1 : voltage regulator bridged (DSP at 5V=extern voltage) bit[2] xpoff : DSP voltage regulator 0 : enable 1 : disable (DSP at 0V) bit[1:0] voltage : DSP/V8-voltage 00 : 3.75V 01 : 3.5V 10 : 3.3V 11 : 3V	V8W_VOLTREG

10.7. DSP EMU-control registers

Table 10–6: V8 DSP-EMU Write Registers

Register Address	Function	Name
DSP-EMU CONTROL REGISTERS		
$b073_{hex}$	**EMU-RAM data high (Reset: 0)** use this register to fill the (external) XDFP-emu-ram bit[7:0] data : xdfp opcode [23:16]	V8W_EDHI
$b074_{hex}$	**EMU-RAM data middle(Reset: 0)** use this register to fill the (external) XDFP-emu-ram bit[7:0] data : xdfp opcode [15:8]	V8W_EDMD
$b075_{hex}$	**EMU-RAM data low(Reset: 0)** use this register to fill the (external) XDFP-emu-ram bit[7:0] data : xdfp opcode [7:0]	V8W_EDLO

$b076_{hex}$	**EMU-RAM configuration / EMU-RAM address high-byte (Reset: 10_{hex})**
	use this register to define the address of the (external) XDFP-emu-ram that should be filled with the data in {V8W_EDHI,V8W_EDMID,V8W_EDLO} and to configure the emu-interface
	bit[7:7] rd; //0.. "normal" EMU mode 1..read back the external ram
	//activates the data line drivers!
	bit[6:6] emuon; //1..XDFP uses ext. EMU-ram 0..XDFP uses its internal rom
	bit[5:5] eadrv; //1..adress-lines to external RAM are driven
	// 0..adress-lines are tristate
	bit[4:4] clkdrv; //1..dfp-clock driven to Pin EMUCLK
	//0..Pin EMUCLK is tristate
	bit[3:0] adrhi : emu-ram address[11:8]
		V8W_EAHI

$b077_{hex}$	**EMU-RAM address low-byte (Reset: 0)** use this register to define the address of the (external) XDFP-emu-ram that should be filled with the data in {V8W_EDHI,V8W_EDMID,V8W_EDLO} bit[7:0] adrlo : emu-ram address[7:0]	V8W_EALO

Table 10–7: V8 DSP-EMU Read/Write Registers

Register Address	Function	Name
DSP EMU CONTROL REGISTERS		
$b06c_{hex}$	**Read back XDFP emu-data, high-byte** bit[7:0]data: contains bit[23:16] of the selected emu-ram address	V8R_EDHI
$b06d_{hex}$	**Read back XDFP emu-data, mid-byte** bit[7:0]data: contains bit[15:8] of the selected emu-ram address	V8R_EDMDI
$b06e_{hex}$	**Read back XDFP emu-data, low-byte** bit[7:0]data: contains bit[7:0] of the selected emu-ram address	V8R_EDLO

11. Glossary

lower ROM: 8k Program ROM form 0000 to 1FFF. Can be shadowed by 8k EMU-ROM

BDT - Buffer Descriptor Table

bootloader: routine in upper rom – handles firmware-startup and code&data download from external EEPROM.

Descriptor:Contains all information of the USB-characteristic. Transferrred to host during enumeration.

FU - Feature Unit

IT – Input Terminal

MU – Mixer Unit

OT – Output Terminal

Section1:256bytes data area. Contains default settings for hardware and audio processing and programmable descriptor components.

Shadow RAM:8k – internal RAM. Shadows the lower ROM. Used for downloading new firmware or emulating firmware before generating a progam ROM.

SU - Selector Unit

upper ROM:4k – Program ROM from 2000 to 2FFF. Can NOT be shadowed

V8 – name of the internal microcontroller

XDFP – name of the internal DSP – core

12. Application Note History

1. Application Note Software: “UAC 357xB Programmer’s Guide”, ..., 6251-5XX-XAS. First release of the application note software.

Micronas GmbH
Hans-Bunte-Strasse 19
D-79108 Freiburg (Germany)
P.O. Box 840
D-79008 Freiburg (Germany)
Tel. +49-761-517-0
Fax +49-761-517-2174
E-mail: docservice@micronas.com
Internet: www.micronas.com

Printed in Germany
Order No. 6251-5XX-XAS