DV Dongle Technical Reference

Ver. 1.00

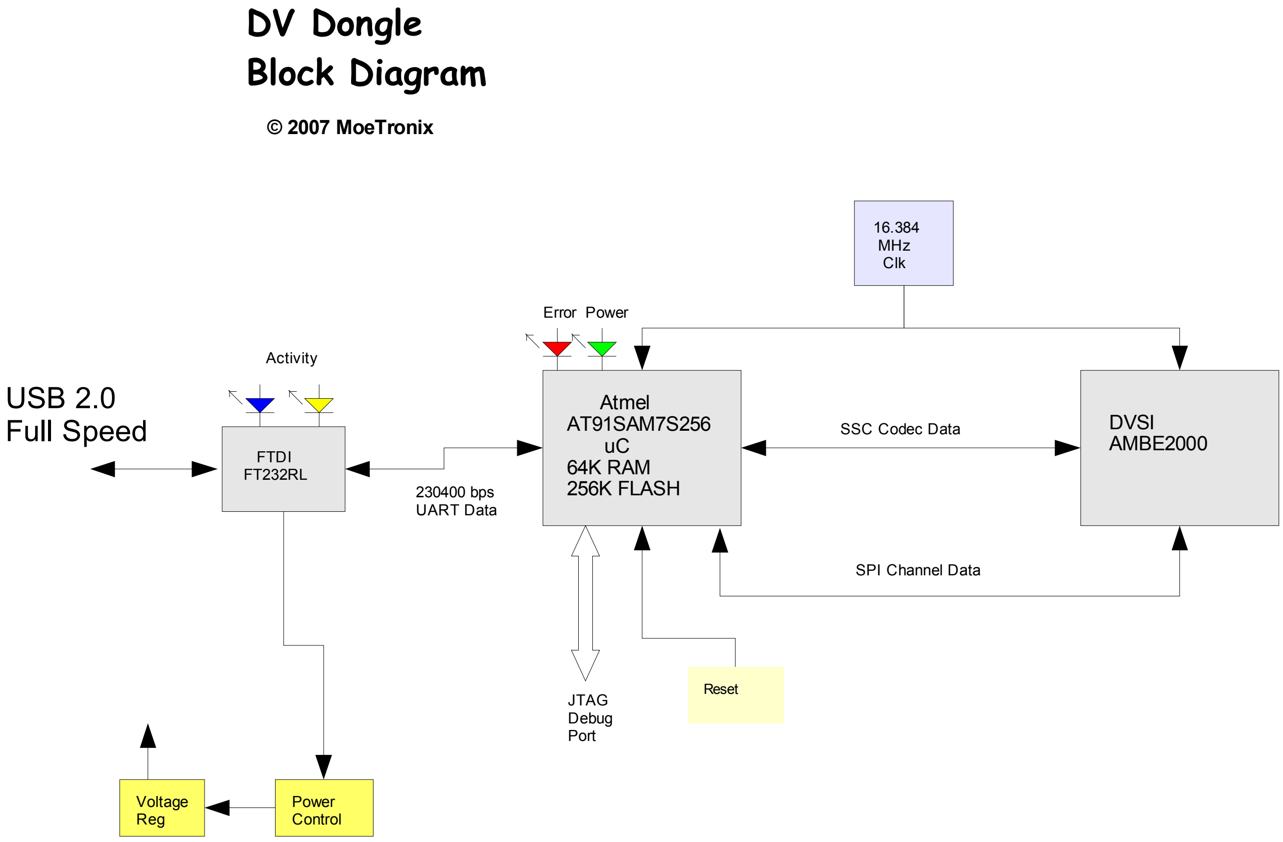
Nov 30, 2007

MoeTronix
www.moetronix.com

Table of Contents

1. DVDongle Architecture

1.1. *Block Diagram*



1.2. *Functionality*

The DVDongle consists of an FTDI FT232RL USB to virtual serial interface chip connected to an Atmel AT91SAM7S256 ARM7 Microcontroller which is used to communicate with a DVSI AMBE2000 voice compression/decompression chip. The AMBE2000 chip provides full duplex operation which means there are four simultaneous data streams passing to and from the DV Dongle and the host. Compressed data in and out and uncompressed audio data in and out are formatted into ASCP data packets and are transferred over the USB virtual serial port interface.

1.3. *FTDI USB interface*

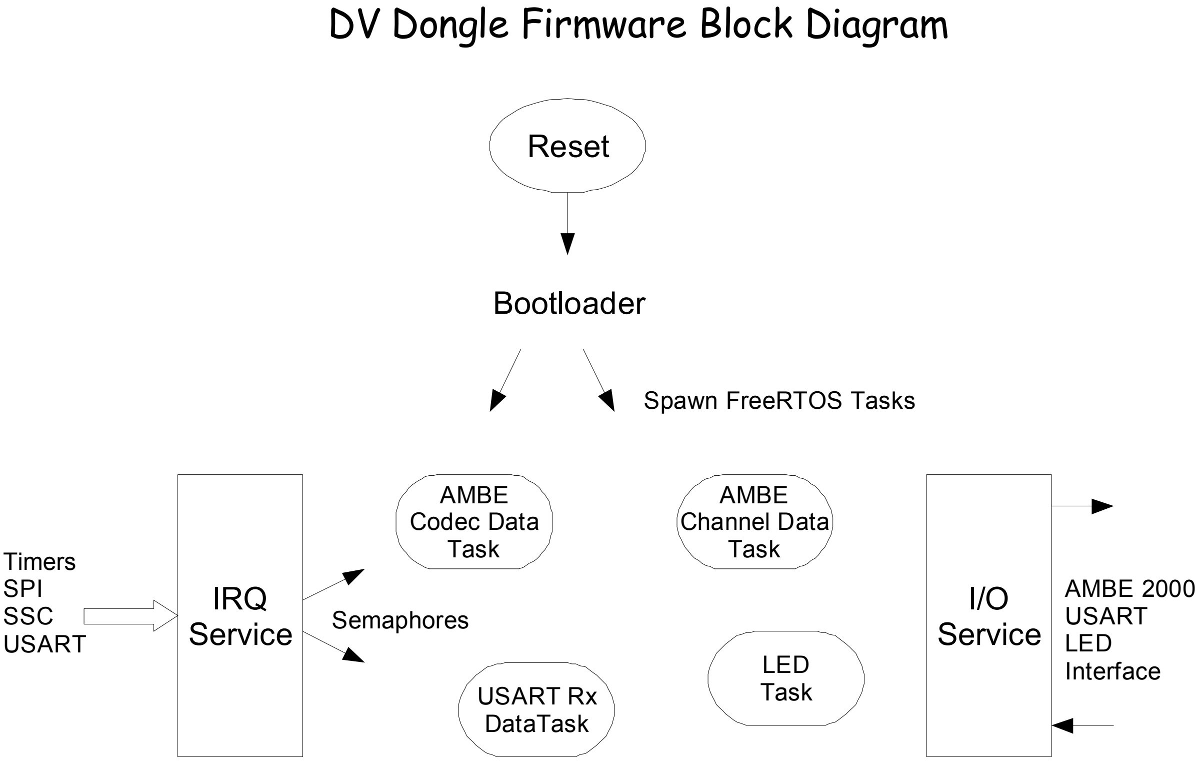
The USB interface of the DVDongle is provided by an interface chip manufactured by FTDI http://www.ftdichip.com/ The chip implements the low level USB protocol stack in hardware and provides the basic data connection to the DVDongle by making the USB port look like a satandard RS232 serial port. This allows the PC host interface to use standard drivers and API calls to talk to the DVDongle.

FTDI provides a Windows driver for the FT232RL chip and so the low level USB interface is hidden from the application software writer. The DVDongle's virtual serial port must be set to 230400 baud with 8 bits data and no parity and no hardware handshaking.

2. Firmware Description

2.1. Overview

2.1.1.Block Diagram



2.1.2.Boot Loader

Upon Reset, the Bootloader flash code section is mapped at 0x00000000 and so this block is executed first. The first thing the bootloader does is copy itself into RAM and then begin executing out of RAM. This is required in order to be able to self program the flash memory while executing code.

The ARM7 “REMAP” feature is used to change the execution from Flash to RAM.

No interrupts are used in the bootloader code as the Usart is polled in the main loop. The bootloader checks the checksum of the Application flash code and if it passes, then the processor jumps directly into the main application code. If the checksum fails or if the testpoint is held low during reset, then the CPU stays in the bootloader code awaiting commands to program the flash wth new application code.

2.1.3.Application Firmware

The application flash code resides just above the 16K bootloader block. It is written as if the normal reset and IRQ vectors are located at 0x4000 instead of the normal 0x0000. The bootloader copies the exception code block into ram before jumping into the application code so the physical translation is transparent to the application writer.

The application code makes use of the on chip DMA capability to make the movement of data much more efficient between the peripherals and memory. The USART talks to the FTDI chip. The SSC is used to transfer the 16 bit PCM 8000 sps data streams to and from the AMBE2000 chip. The SPI port is used to transfer the 48 Byte compressed channel data frames to and from the AMBE2000 chip.

Various Timers are used to manage the AMBE frame timing and ancillary timing tasks.

The FreeRTOS realtime operating system is used for overall task management and signaling. All the code except for the C Startup code is written in C and compiled with the GNU Arm compiler/linker.

2.2. Source Code Organization

The source code is divided into two independent and separate projects, the bootloader(“DVDongleBoot”) and the main application code(“DVDongle”). The Bootloader project is small and does not use the FreeRTOS so all the source files are in one flat directory.

The application code is divided into sub folders.
The FreeRTOS folder maintains the same folder structure as the original FreeRTOS source distribution.
The Source folder contains all the *.c, *.s source files.
The Include folder contains all the *.h header files.
The root folder contains the makefile, linker script, and the compiler/linker output files.

2.3. Development Tools

2.3.1.Eclipse IDE

While not necessary, the Eclipse IDE was used for project organization, editing, compiling, and debugging. An excellent tutorial and setup guide, “Using Open Source Tools for AT91SAM7S Cross Development“ has been written by James P. Lynch that can be used to guide one through the installation and setup of the Eclipse and GNU arm tools chain on a Windows based platform. All the tools will of course run natively on a Linux box as well.

The tutorial can be found here or if it has moved just Google for the title.
http://www2.amontec.com/sdk4arm/ext/jlynch-tutorial-20061124.pdf

2.3.2.GNU Compiler

The GNU ARM compiler chain was used to generate the final object code to be loaded into the DVDongle.

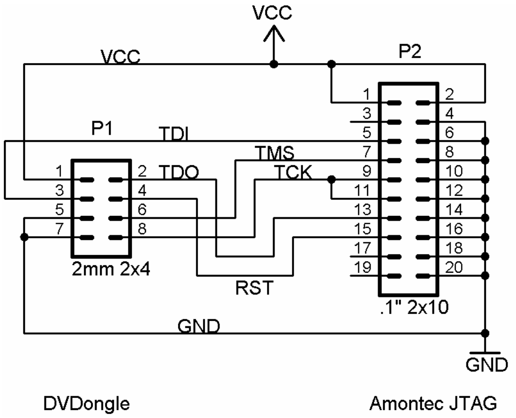
A precompiled Windows GNU ARM tool chain can be found here: http://www.yagarto.de/

2.3.3.Amontec JTAG Debugger

A fairly low cost JTAG debugger can be found here: http://www.amontec.com/

This is only really needed to initially program the bootloader code onto the DVDongle. If you are only modifying the application code, then one can just download using the bootloader and the USB virtual COM port.

An adapter cable will be needed to go from the standard 20pin JTAG connector to the DVDongle’s 8pin 2mm connector. A schematic of the pin out conversion is shown below.



2.3.4.Flash32Loader

Flash32Loader is a utility program that can be used to update the application firmware inside the DVdongle. It takes a .hex file that is generated by the GNU compiler and loads it into the DVDongle flash over the USB virtual COM port interface. This allows code updates without the need for any JTAG hardware or taking the DVDongle apart.

3. Basic Protocol Concepts

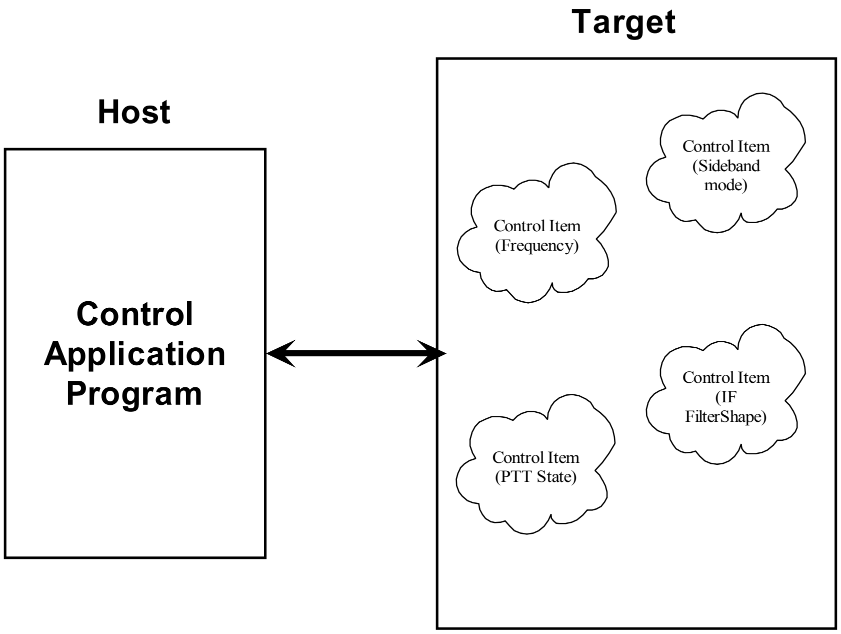
This specification describes the protocol used to communicate with the DVDongle USB digital voice compression/decompression device. The protocol used is based on a more generic protocol called ASCP(Amateur Station Control Protocol).

Definitions used in this specification:
In the case of the DVDongle, the host is the PC and the Target device is the DVDongle USB hardware.

Host == The main initiator of communications. Typically would be a PC or other computer system such as a custom user interface controller.

Target == The device that is to be controlled or monitored by the Host.

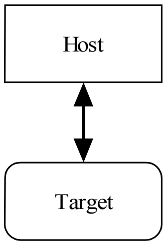
Control Item == The value, setting, or state of the target that is to be controlled or monitored by the Host. For example Frequency, Antenna direction, modulation mode, transmitter state, etc.

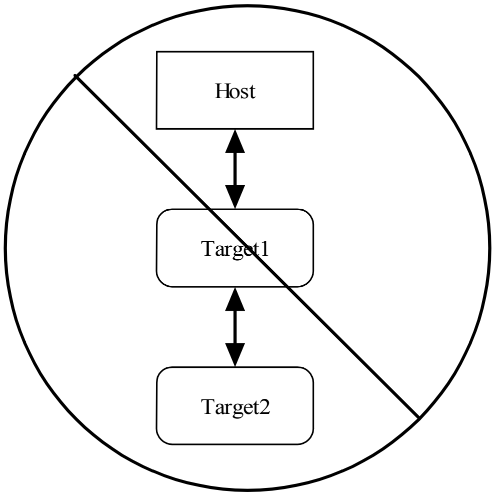

Target
Host
Control Application Program
Control Item (Sideband mode)
Control Item (Frequency)
Control Item (IF FilterShape)
Control Item (PTT State)

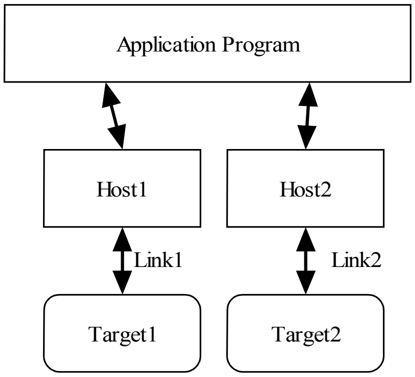
Data Item == Digital data associated with the received or transmitted signal. This could be digitized audio, IF, frequency domain data(FFT), information data such as text or AX25 data, etc.

Message Block == A contiguous block of bytes comprising a single Control Item or Data Item transfer from target to host or host to target.

To simplify the protocol, the link can only comprise one host and one target. The Host is the only one that can set or request Control items. This means a Target device cannot connect to another target device or daisy chain to other targets. The Host can control multiple Targets by utilizing multiple links such as USB endpoints or multiple TCP/IP Sockets.







The protocol allows a Target to send unsolicited Control Item messages to the Host. This is desirable for updating the Host when a something changes in the Target without the need for polling by the Host. An example would be when a user changes the frequency of the radio using the radio's frequency knob. The target can send the updated frequency as it occurs without requiring the Host to ask for it.

Message blocks contain the block length in the message header. This is useful to aid in decoding messages as well as being able to support variable length Control Items. For example, a Control Item containing the text string for the Target's manufacturer and model number can be different lengths.

The protocol contains a mechanism for obtaining information about the range and resolution of a Control Item. This allows the Host to obtain the limits of Control Items for the particular target device. For example, the frequency ranges of the target as well as the frequency step size can be obtained without any knowledge of the model or manufacturer of the radio. This capability is key in being able to write a generic application interface program without having to maintain a list of all possible radios and devices that may be connected.

Target devices are not required to implement all the functionality of the protocol. A simple Target device that turns on and off a light need only implement the single Control Item message associated with it. This allows micro controllers with limited capability to be used.

The Data Item message blocks allow various raw data blocks to be sent and received along with the Control Items over the same physical link. The Header type allows up to 4 logical channels of data to be specified in each direction. This permits sending digitized audio, digitized I/Q IF data, etc. to and from a target over the same physical connection.

Note that there is no synchronization or error handling mechanism in this protocol. This layer of protocol assumes that the block synchronization and error handling is done at a lower level. This is a reasonable assumption since Ethernet, USB, IEEE 1394, and most other modern physical links provide error recovery. The exception is RS232 serial links. This protocol will work over it as-is but if any data corruption is anticipated, then some simple method such as "SLIP" could be used to frame the data bytes and perhaps add error detection as well. A little "sanity" checking of message block parameters can be used as a simple error detection scheme and allow the RS232 system to eventually sync back up. Adding a timeout will handle the case where a link is disconnected in the middle of a transfer. Since RS232 serial links are being phased out, it doesn't make sense to burden a protocol to handle such a legacy physical link.

4. Message Block Format

The basic message structure starts with a 16 bit header that contains the length of the block in bytes and also a 3 bit type field. If the message is a Control Item, then a 16 bit Control Item code follows the header and contains the code describing the object of the message block. This is followed by an optional number of parameter or data bytes associated with this message. The byte order for all fields greater than 8 bits is "Little Endian" or least significant byte first.

Control Item Message block format:

16 bit Header(lsb msb)	16 bit Control Item(lsb msb)	Parameter Bytes

Data Item Message block format:

16 bit Header(lsb msb)	N-Data Bytes

The 16 bit header is defined as follows:

8 bit Length lsb	3 bit type	5 bit Length msb

The 13 bit Length parameter value is the total number of bytes in the message including this header. The range of the message Length is 0 to 8191 bytes.

A special case for Data Items is that a message length of Zero is used to specify an actual message length of 8194 bytes(8192 data bytes + 2 header bytes). This allows a 8192 byte data block (a power of 2) to be used which is useful in dealing with FFT data.

The message type field is used by the receiving side to determine how to process this message block. It has a different meaning depending upon whether the message is from the Host or Target.

3 bit Msg Type field	Message Source	Message Type
000	Host	Set Control Item
001	Host	Request Current Control Item
010	Host	Request Control Item Range
011	Host	Data Item ACK from Host to Target
100	Host	Host Data Item 0
101	Host	Host Data Item 1
110	Host	Host Data Item 2
111	Host	Host Data Item 3
000	Target	Response to Set or Request Current Control Item
001	Target	Unsolicited Control Item
010	Target	Response to Request Control Item Range
011	Target	Data Item ACK from Target to Host
100	Target	Target Data Item 0
101	Target	Target Data Item 1
110	Target	Target Data Item 2
111	Target	Target Data Item 3

4.1. Detailed Description of the Message Block Types and Their Purpose

4.1.1.Set Control Item

This Message type is sent from the Host to the Target requesting that the Target change the specified Control Item to the new value supplied in this message. A request to change to a new frequency would be an example of this type of message. The Target must respond to this message.

4.1.2.Request Control Item

This Message type is sent from the Host to the Target requesting that the Target respond with its current state or value of the specified Control Item of this message. A request to get the current S-meter reading would be an example of this message type. The Target must respond to this message.

4.1.3.Request Control Item Range

This Message type is sent from the Host to the Target requesting that the Target respond with the acceptable range of values of the Control Item supplied in this message. A request for the targets frequency range(s) and step sizes would be an example of this message type. The Target must respond to this message.

4.1.4.Response to Set or Request Current Control Item

This Message type is sent from the Target to the Host in response to a request from the Host to either set or just return the current value of the Control Item supplied in this message. This message contains the current value of the Control Item. It is sent in response to either the "Set Control Item" or "Request Control Item" message.

4.1.5.Unsolicited Control Item

This Message type can be sent from the Target to the Host without any request from the Host. It contains the current value of the Control Item supplied in this message. This message can be sent at any time to the Host. It can be used to update the Host to any changes that have occurred in the Target Control Items. An example would be if the user changed frequency using the Targets frequency knob, then the Target could send the new Control Item value to the Host without having to wait for the Host to ask for it. There is no response back from the Host when this message is received.

4.1.6.Response to Request Control Item Range

This Message is sent from the Target to the Host in response to a "Request Control Item Range" message from the Host. It contains the allowable range and step size of the Control Item supplied in this message.

4.1.7.Data Item Messages

Data Item message allow data messages to be allocated to different logical "channels". Different types of data blocks may be interleaved together and this mechanism allows each end to keep the data separated. For example, Data Item 2 blocks may be digitized audio from a Target receiver that needs to be processed and sent to a soundcard speaker. Data Item 3 Blocks may be spectral data from an FFT inside the Target receiver that needs to be sent to the Host applications display screen.

The current scheme allows up to four different logical channels for each data direction.

4.2. The ACK and NAK Messages and Their Purpose

A "NAK" message is a 16 bit header without a Control Item or parameters (Message length of 2) . When the NAK message block is returned by the Target, it indicates that the specified Control Item is not supported. This allows a target to implement only the Control Items it actually needs. Any Host message requesting an unimplemented Control Item will be returned the NAK message. The Host can then exclude this Control Item from its list of Items to control or monitor.

As an example, suppose a Host requests the elevation setting from a rotor Target controller that only supports azimuth readings. The Target controller would just return the NAK header.

Implementation on the Target side is easily done by simply decoding only the Control Item messages that it supports and returning the NAK for all others.

On the Host side, one could initially poll the Target for all the Control Items it may use and then tag the ones that return NAK for exclusion.

This dynamically allocated feature set allows application software to determine on the fly what capabilities are available without any prior knowledge of the Target device.

A Data Item "ACK" message is a 16 bit header with a Message Type = 011b with a single parameter byte specifying the Data Item (0 to 3). The 16 bit header is a fixed value (0000 0011 0110 0000 = 0x0360). The parameter byte following the header specifies which Data Item block that is being ACK'd.

For example if the Target received a block of Data Item 2 data correctly it could send the following back to the Host:
[03][60] [02]

The ACK response messages is to provide handshaking to data item transfers. If a data item message is received correctly then an ACK response message could be sent back to the sender. This implementation is optional as one may want to stream data without error checking or only ACK periodically the data stream.

5. DVDongle Control/Data Item Definitions

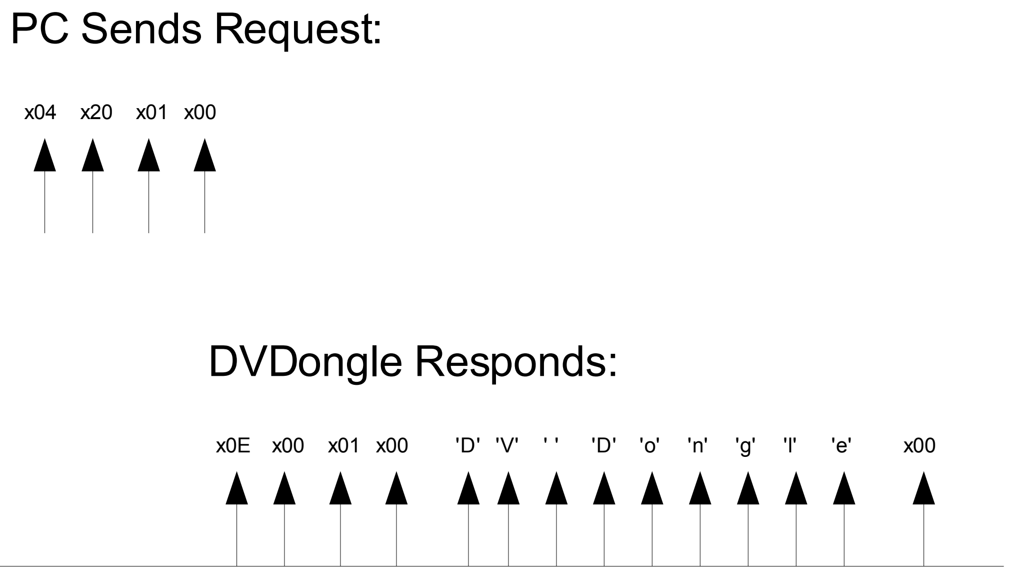
All examples use hexadecimal notation within brackets [] for the individual byte values.
The "Target" referred to is the DVDongle unit.

5.1. General Control Items

5.1.1. Target Name

Purpose: Returns an ASCII string describing the Target device.
Control Item Code : 0x0001
Control Item Parameter Format: The data is a NULL(zero) terminated character byte string.

Example, to request the target name, the host sends:
[04][20] [01][00]
The Target responds with "DV Dongle" :
[0E][00] [01]00] [44][56][20][44][6F][6E][67] [6C] [65] [00]



5.1.2. Target Serial Number

Purpose: Contains an ASCII string containing the Target device serial number.
Control Item Code : 0x0002
Control Item Parameter Format: The data is a NULL(zero) terminated character byte string.

Example, to request the target serial number the host sends:
[04][20] [02][00]
The Target responds with "MT123456" or whatever the serial number of the particular device is:
[0C][00] [02]00] [4D][54][31][32]33][34[35[36][00]

5.1.3.Interface Version

Purpose: Contains the version number of the Host or Targets implemented Interface. This allows the Host or Target to display or adapt to different versions of the interface.
Control Item Code: 0x0003
Control Item Parameter Format: The data is a 2 byte 16 bit unsigned variable equal to the version times 100. For example the value 123 would be version 1.23.

Example, to request the target interface version the host sends:
[04][20] [03][00]
The Target with an interface version of 5.29 responds with:
[06][00] [03]00] [11][02]

5.1.4.Hardware/Firmware Versions

Purpose: Contains the Firmware or Hardware version information of the Target.
Control Item Code: 0x0004
Control Item Parameter Format:
The first parameter is a 1 byte Firmware ID specifying which firmware or hardware version to retrieve.
ID=0 returns the ARM7 boot code version.
ID=1 returns the ARM7 firmware version.

The version data is a 2 byte 16 bit unsigned variable equal to the version times 100. For example the value 123 would be version 1.23.

Example, to request the DVDongle firmware version host sends:
[05][20] [04][00] [01]
The Target with an DVDongle firmware version of 5.28 responds with:
[07][00] [04]00] [01] [10][02]

Example, to request the DVDongle boot code version host sends:
[05][20] [04][00] [00]
The Target with a DVDongle firmware version of 5.29 responds with:
[07][00] [04]00] [00] [11][02]

5.1.5.Status/Error Code

Purpose: Contains the Error/Status code(s) of the Target. This item is used to notify the Host of any error or problem using a list of code values. Once the error code(s) are obtained, the host can interrogate the Target "Error String" Control Item to obtain a description string of the error(s) or status.
Control Item Code: 0x0005
Control Item Parameter Format: The data is a list of 1 byte unsigned variable equal to the error number associated with a particular error. There can be multiple error codes returned by the Target.

0x00 = Stopped
0x01 = Running
0x0E = Boot mode Idle
0x0F = Boot mode busy programming
0x80 = Boot mode programming error

Example, host request status:
[04][20] [05][00]
The stopped Target responds with
[05][00] [05]00] [00]

5.2. DVDongle Specific Control Items

5.2.1. Run State

<u>Purpose:</u> Controls the operational state of the DVDongle.
<u>Control Item Code:</u> 0x0018
<u>Control Item Parameter Format:</u>

The first parameter is a 1 byte value defined as:
0x01 = Stop
0x02 = Run.

Example: Request to start the DVDongle running:
The host sends:
[05][00] [18][00] [01]
The Target responds with:
[05][00] [18][00] [01]

5.2.2. DVDongle Setup (*Not Yet Implemented*)

<u>Purpose:</u> Controls the setup of the DVDongle parameters.
<u>Control Item Code:</u> 0x0034
<u>Control Item Parameter Format:</u>

The First parameter defines the compressed data format and is a 1 byte value defined as:
0x00 = AMBE raw channel data frame mode(default)
0x01 = D-Star Data Mode.
The Second parameter byte is compressed data FIFO size(2 to 32) (default 16)
The Third parameter byte is uncompressed codec data FIFO size(2 to 32) (default 16)

Example: Request to setup the DVDongle for AMBE raw mode with FIFO sizes of 8:
The host sends:
[07][00] [34][00] [00] [08] [08]
The Target responds with:
[07][00] [34][00] [00] [08] [08]

5.3. DVDongle Data Item Definitions

5.3.1. DVDongle CODEC (Uncompressed audio data) Data Item 0

<u>Purpose:</u> This is the uncompressed audio data item message that is sent/received by the host when the DVDongle is running.
<u>Data 0 Item Parameter Format:</u>
All data 0 blocks to/from the DVDongle are fixed size of 160, 16 bit samples containing 320 bytes of data with the 2 byte data item 0 header.

0x42	0x81	320 Data Bytes

The 320 data bytes represent 160, 16 bit sample values representing the 8000sps PCM uncompressed audio.

The byte breakdown for the data:

S1 lsb	S1 msb	S2 lsb	S2 msb	……..	S160 msb	S160 lsb

5.3.2.DVDongle Channel(Compressed) Data Item 1

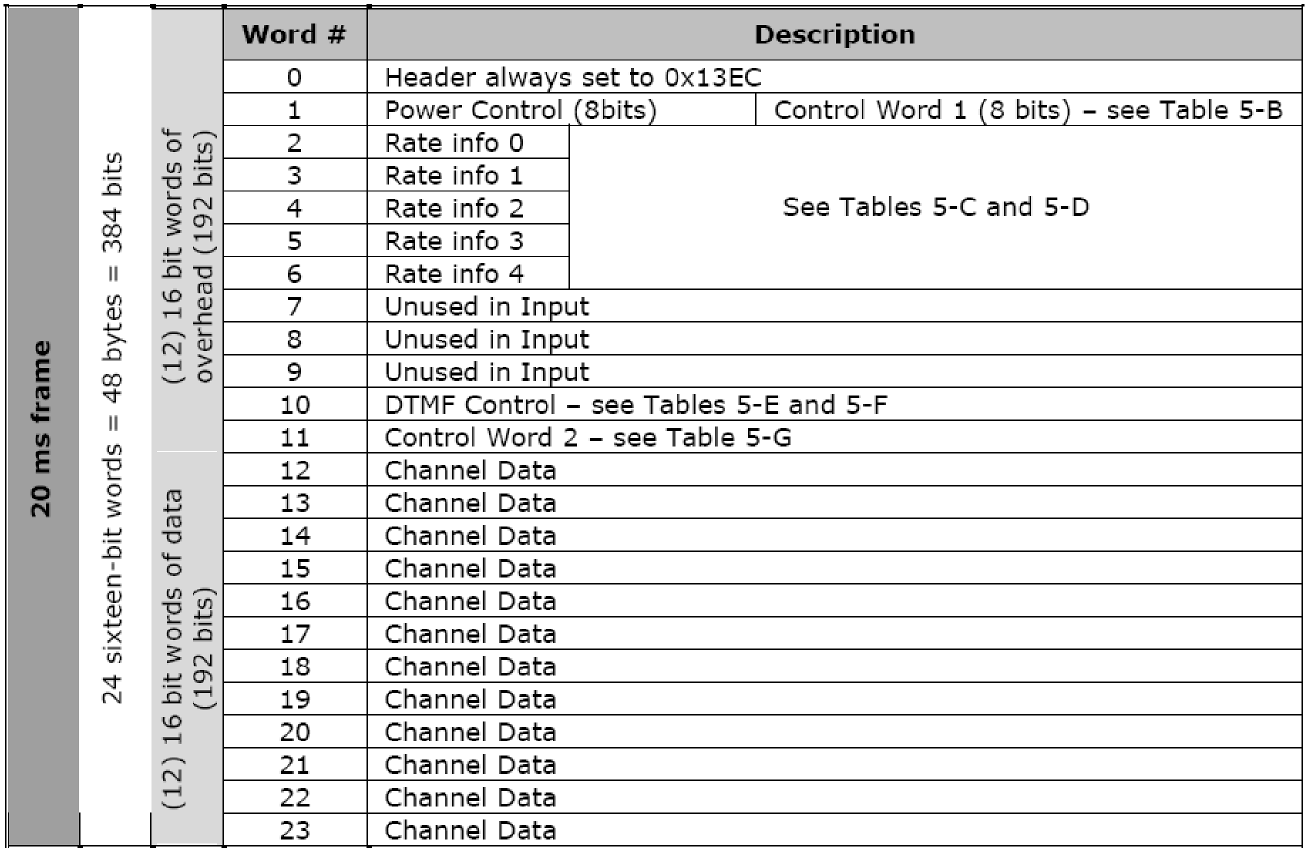
Purpose: This is the compressed audio data item message that is sent/received by the host when the DVDongle is running.

Data 1 Item Parameter Format:

The format of this message depends on the DVDongle setup mode that is selected. For the AMBE raw data mode, all 48 of the AMBE channel data frame bytes are sent and received in each data packet.

0x32	0xA0	48 Data Bytes

The Twenty Four, 16 bit data fields are defined below and detailed information for all the parameters can be obtained from the DVSI AMBE2000 data sheet.



			Word #	Description	
20 ms frame	24 sixteen-bit words = 48 bytes = 384 bits	(12) 16 bit words of overhead (192 bits)	0	Header always set to 0x13EC	
			1	Power Control (8bits)	Control Word 1 (8 bits) – see Table 5-B
			2	Rate info 0	See Tables 5-C and 5-D
			3	Rate info 1	
			4	Rate info 2	
			5	Rate info 3	
			6	Rate info 4	
			7	Unused in Input	
			8	Unused in Input	
			9	Unused in Input	
			10	DTMF Control – see Tables 5-E and 5-F	
			11	Control Word 2 – see Table 5-G	
		(12) 16 bit words of data (192 bits)	12	Channel Data	
			13	Channel Data	
			14	Channel Data	
			15	Channel Data	
			16	Channel Data	
			17	Channel Data	
			18	Channel Data	
			19	Channel Data	
			20	Channel Data	
			21	Channel Data	
			22	Channel Data	
			23	Channel Data	

5.4. DVDongle Software/Firmware Update Item Definitions

This set of items is used to update the DVDongle firmware that resides in the Atmel AT91SAM7S processor.

5.4.1.Update Mode Control

Purpose: Controls the Updating of Software or Firmware code.

Control Item Code: 0x0300

Control Item Parameter Format:

The first parameter is a 1 byte device ID (0 to 255). This parameter is ignored.

Parameter 2 is a 1 byte the Mode command.
0x00 == ENTER (Enter boot loader code if it is not running in it currently)
0x01 == START (Begin the update process)
0x02 == END (End update process and jump back into user code)
0x03 == ABORT (Abort update process)

Parameter 3,4,5, and 6 is a 4 byte password to protect against inadvertent programming.
The DVDongle password is:
Parameter 3 == 0x5A
Parameter 4 == 0xFF
Parameter 5 == 0xA5
Parameter 6 == 0x00

Example, to start the update process:
The host sends this:
[0A][00] [00][03] [00] [01] [5A] [FF] [A5] [00]
The Target responds with:
[0A][00] [00][03] [00] [01] [5A] [FF] [A5] [00]

5.4.2.Update Mode Parameters

Purpose: Request programming parameters from the target device.
Control Item Code: 0x0302
Control Item Parameter Format:
The first parameter is a 1 byte device ID (0 to 255). This parameter is ignored.

The response contains these additional bytes:
Parameter 2,3,4,5 is the Flash size in bytes (32 bit unsigned integer LSB first)

Parameter 6,7,8,9 is the Flash programming page size in bytes (32 bit unsigned integer LSB first)

Parameter 10,11,12,13 is the Flash Sector size in bytes (32 bit unsigned integer LSB first)

The host sends this to request the programming parameters from the target device 0:
[05][20] [02][03] [00]
The Target responds with: (256K byte FLASH, 256 byte page, 16384 byte sector)
[11][00] [02][03] [00] [00] [00] [04] [00] [00] [01] [00] [00] [00] [40] [00] [00]

5.4.3.Update Mode Data Item 0 Block

Purpose: This is the main data item message that is sent to the target containing device programming data.
Data 0 Item Parameter Format:

Data blocks sent to the DVDongle from the host are a fixed size containing a 4 byte address, followed by "Flash Programming Page Size" bytes of data with the 2 byte data item 0 header.

Hdr1	Hdr2	4 byte Starting address for this data block	"Flash Programming Page Size" data Bytes

Example, to program a block of data at address 0x12345 in the DVDongle flash. Assume the flash programming page size is 256.

The host sends this: 2 hdr bytes + 4 address bytes + 256 data bytes = 262 bytes
[06][81] [45] [23] [01] [00]   [9A] [78] [56] [34] [12] ....[37]

The DVDongle responds after programming the block into flash with:
[03][60] [00]

The host can now send another data block after receiving the ack message from the target.

6. PC Software Interface

6.1. Basic Tasks

The Host PC software must perform several tasks in order to use the DV Dongle for voice compression and de-compression.

- Perform low level COM Port I/O to send and receive data packets to and from the DV Dongle.
- Maintain correct data flow in and out of the DV Dongle.
  Every 20mSecs the DV Dongle will send a channel and codec data packet and also expects to receive a channel and codec data packet every 20mSecs. There are buffers in the dongle so one need only keep a one-to one correspondence to data packets going into the dongle as are coming out.
- Perform 8000 sps 16 bit monophonic PCM Sound Card I/O if a soundcard is being used.
- Format the DV Dongle Channel Data packets in the correct AMBE 2000 frame format.
- Provide basic status and start/stop functionality.

6.2. Streaming Data Issues

The DV-Dongle presents some different challenges to the programmer in that it requires data to be streamed into it and taken from it at a fixed data rate. This is because the AMBE 2000 chip is designed for real time only voice compression/de-compression and so must run at its fixed data rates. So unlike compressing a file where you just read and write data as fast as you can process it, the DV Dongle must be fed at its data rate.

6.2.1. DV Dongle Data Packet Timing

The best approach to interfacing is to send a data packet to the dongle for every one you receive from the dongle. Even though the PC timing may be very sloppy and the actual packet timing may vary widely, by doing this one maintains the correct overall average timing and the dongle will manage the exact timing required by the AMBE 2000 chip. If the dongle gets out of sync, it will flash the Red LED indicating a data overflow or underflow condition.

6.2.2. Sound Card and DV Dongle Data Rate Differences

This is perhaps the most difficult issue to deal with when using the dongle and PC sound card. Even though both are setup to run at 8000 sps, due to the fact that the sound card and dongle are clocked using different oscillators, there will be a small difference in actual data rates between the two devices.

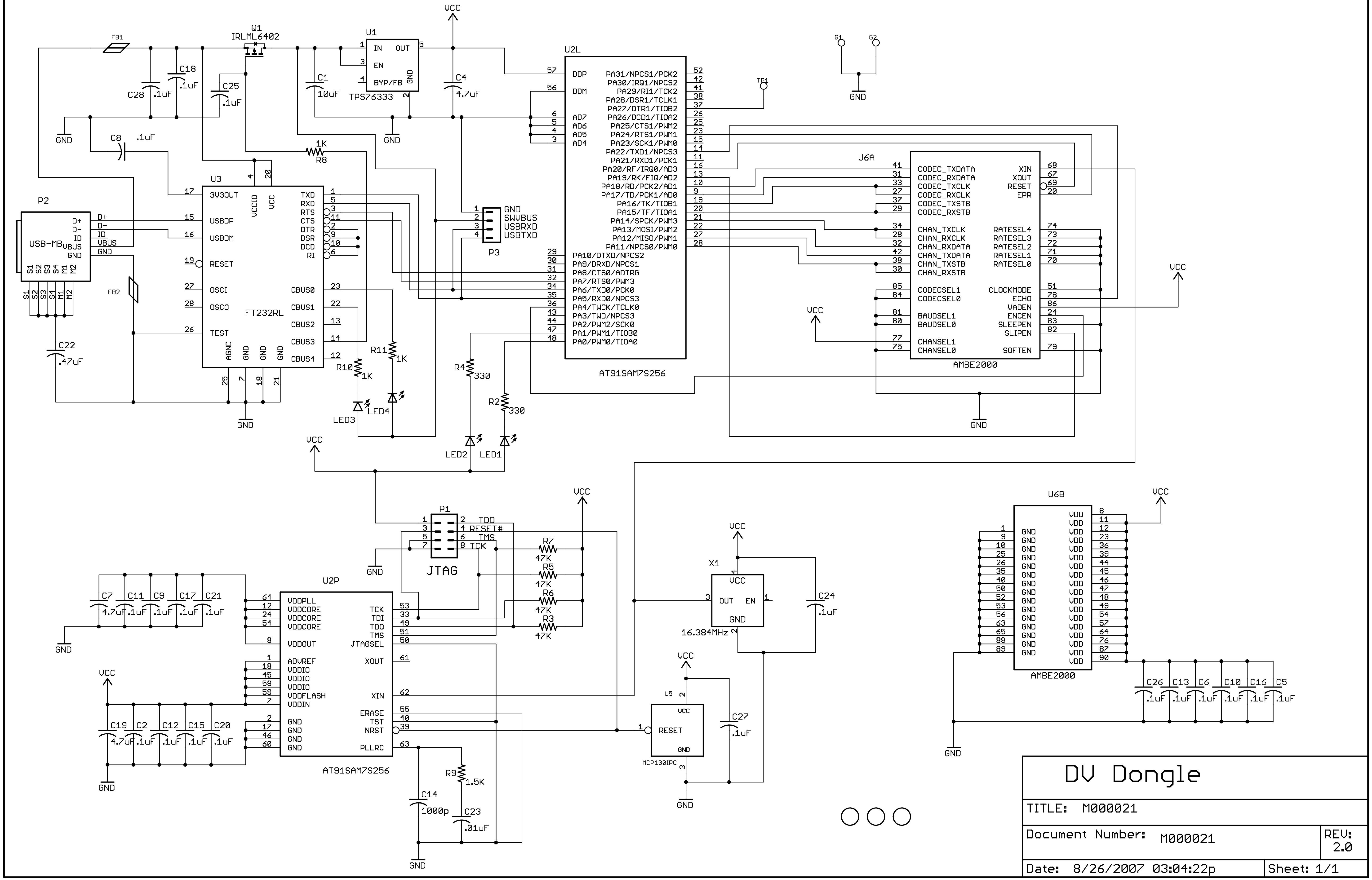
If the sound card is running slightly faster than the dongle, then at some point, one must discard some data going to the dongle. This will cause a disturbance in the sound. Likewise if the sound card is running slower then one must fill in some data going to the dongle to keep it from under flowing.

Normally the difference in data rates is in tens of parts per million so the disturbance does not happen too often and so one can probably live with a blip every few minutes. The problem is more severe with Windows operating systems because Windows has something called Kmixer that is a system software module that re-samples the users data into some other sample rate before going to the sound card. The problem with this is that it does not always accurately emulate 8000 sps but may turn it into say 8050 sps. This causes the over/under flow problem to be worse.

One way to get around this is to up/down sample the dongle codec data to 48000 sps before sending it to the sound card. It seems that Windows does a better job converting that rate to the actual sound card than 8000 sps.

6.2.3. D-Star Voice Packet and DV Dongle Data Rate Differences

The same issues exist on the compressed data side. Data coming and going to a D-Star radio(directly or over the Internet) will not be at exactly the same rate as the DV Dongle. The same idea can be applied where one discards a packet or inserts a packet to make up for the periodic rate slips. One could try a silence packet or maybe repeat the last packet. Some experimenting will be required to see which approach sounds better.



DV Dongle

TITLE: M000021

Document Number: M000021

REV: 2.0

Date: 8/26/2007 03:04:22p

Sheet: 1/1