Creating and Adding Sound Files to a PIC24F

By Ian Pearson,

Field Application Engineer, Microchip Europe.

This document discusses the method and steps required to create sample sounds and convert them to the format required for playback on the Microchip Explorer-16 development board and Speech/Audio PICtail™ Plus. To provide an application context we will use the temperature sensor supplied on the Explorer-16, combined with the PIC24F and Speech PICtail Plus to create a speaking thermometer. We will discuss the tools required to sample, manipulate and create a compressed sound file, plus methods to test and debug the design in both the software and hardware domains. Rather than discuss audio compression techniques.

While many advanced audio compression algorithms exist, we will be concentrating on a relatively simple method that requires minimal processing power and achieves good reproduction quality – the IMA-ADPCM (Adaptive Differential Pulse Coded Modulation) algorithm. This is suitable for a standard microcontroller and requires few resources, unlike some of the more advanced options. However, a trade-off exists between resource usage, reproduction quality and processing power – ADPCM provides a suitable balance for embedded systems with moderate playback needs.

More details of the IMA-ADPCM algorithm can be found in Microchip Application Note AN643 – Adaptive Differential Pulse Coded Modulation using PIC® microcontrollers.

It is assumed that the user has an operational knowledge of MPLAB® and the steps required to add files and build a project. The user does not need knowledge of the C programming language to be able to perform the steps defined to create and add new speech samples. However, if the additional excercises are to be performed, the user will need to be proficient in C.

Tools for the job

Sound editor

A PC-based tool is required to record sound and perform post-sample manipulation. This may also allow the user to remove noise, add effects or manipulate the base data prior to saving in the required format and manipulating for use by the PIC24F running the ADPCM algorithm.

Many downloadable tools allow sound/audio to be manipulated. However, the tool chosen must be able to perform two key operations:

– Sound is recorded on a PC at 44.1.kHz stereo or mono and is saved as a standard signed 16bit PCM .WAV file format.

– To play back on our system using the settings provided in the source code we need to convert the .WAV file into an 8kHz mono unsigned 16bit little endian PCM .RAW file. Note however that RAW may have a .RAW, .SND or other file extension depending on the audio editor chosen.

Therefore our sound editor must be able to:

– Downconvert or re-sample the original to 8kHz

– Save a file as PCM unsigned 16bit, little-endian, mono, raw file format.

The editor chosen to perform this was Goldwave. Many other candidates for audio file manipulation exist. However this was suitable for the requirements, provided a demo capability and is relatively inexpensive, less than US$50, to purchase a full version if ultimately chosen for general usage. Critically, it could save raw files in the required format.

For this playback-only application we will be implementing the IMA-ADPCM algorithm. The implementation used provides a 4:1 compression ratio while maintaining a fatihful representation of the original. It should also be considered that many factors in the system, including the output filter, amplifier and compression method used, can contribute to audible errors. Noise and other unwanted artefacts present in the original recording will also be reproduced and may be enhanced during the compression/decompression and reproduction phases. While we are unable to reproduce recording studio conditions for record or playback, we can discuss the methodology to create a system design.

Sound reproduction

To reproduce the original audio from the compressed ADPCM format, a PWM based DAC is created using the Output Compare capability of the PIC24F with an external filter. In this case a 4th order active filter is used to remove aretefacts such as the PWM switching frequency and create a low pass filter to limit the bandwidth to the frequency range of interest (300Hz to 3300Hz). The filter has an approximately 4kHz cut-off point. For more details of the PIC24F Output Compare module please refer to the PIC24F Family Reference Manual.

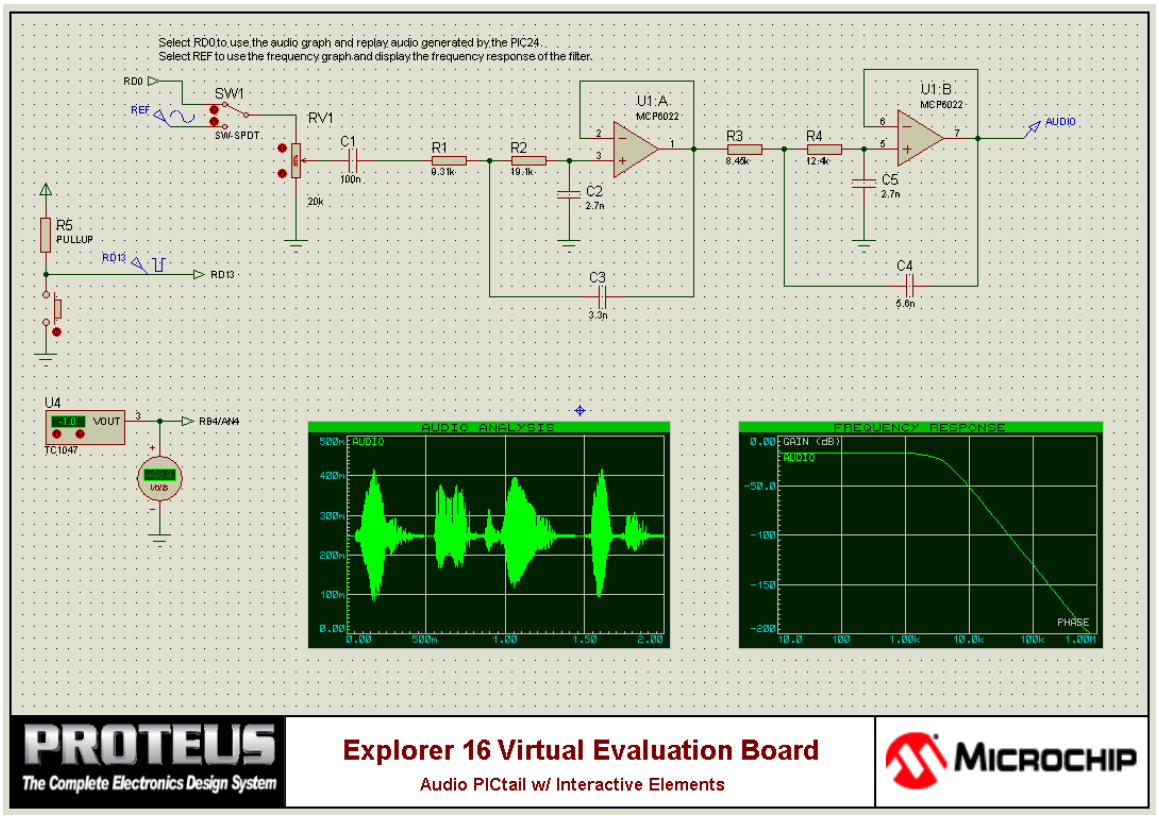

Fig 1: Speech/audio PICtail simulation in Proteus VSM

To speed the design of a suitable output filter we performed simulations with the Proteus VSM plug-in for MPLAB. For this, a design using PIC24F and the Speech/Audio PICtail board was modelled. This allowed us to perform modifications to passives, and hence the filter characteristics, plot the filter response and perform a capture of the audio output for playback and analysis. This system simulation approach affords several productivity gains relative to the hardware build-and-test method. It allows us to perform iterative modification, comparision and analysis quickly and easily and also enables us to debug our code and perfect the temperature conversion algorithm through interactive system simulation. The aim of this process is to reduce the number of physical hardware iterations that might otherwise be needed, thus reducing the overall design and development time.

File manipulation

Once the base sound sample has been saved in the raw format we must perform two additional steps:

– Convert the data into a suitable ADPCM format for playback with the PIC24F source code, using the Winspeech utility

– Create an assembler source file containing the compressed ADPCM sound data in the MPFS file system format, using the MPFSv2 utility.

Winspeech

Winspeech is a utility to convert a raw unsigned 16bit little-endian sound file into an ADPCM compressed sound file compatible with the IMA-ADPCM algorithm implemented in the PIC24F. It can also decompress files to allow us to analyse a file recorded by the PIC24F where the firmware has been enabled to support this. Winspeech is a Microchip utility which is included as part of the example code support package for this article.

Winspeech:

– provides a simple graphical interface to allow a source file of .raw type format to be selected

– uses the IMA ADPCM algorithm, as described in AN643, to create an ADPCM encoded compressed file with a .DAT extension.

The .DAT extension is used because this is the file extension expected to be seen in the example PIC24F source code, where the filename is generated prior to calling an MPFS file system routine. The extension may be chosen by the user, however in this example we have used .dat as the format. Whichever extension is chosen, the 8.3 filename used must match the filename called in the firmware.

Using Winspeech

The graphical inteface provided by Winspeech is shown in Fig.2.

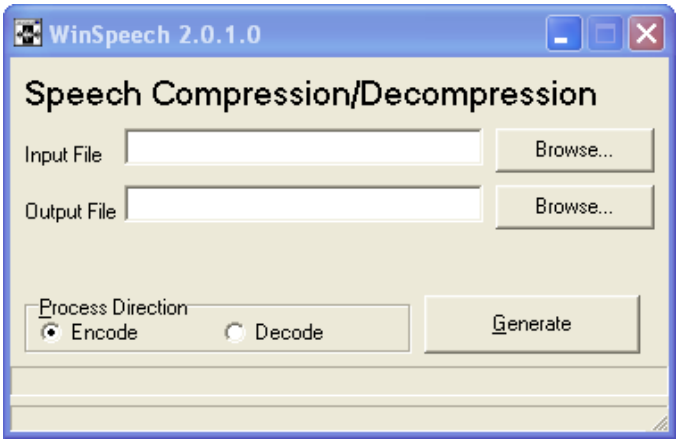

Fig 2: Winspeech utility

We suggest that each stage of the recording and manipulation process is output to its own folder. This will make file handling simpler. The file structure in the demo code can be used an example.

Using the browse button for the input file, navigate to the folder containg the raw file to be converted

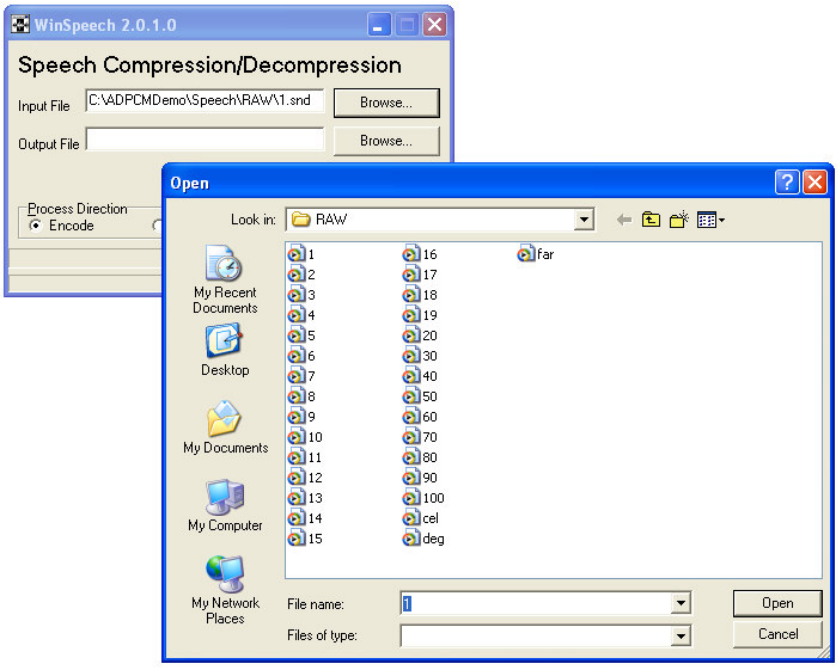

Fig 3: Browse for input file

Using the browse button for the output file, navigate to the folder which will contain the converted .dat file

We suggest that the filenames be kept the same for both the raw and .dat files to allow change tracking. Also, to meet the limitation of the mpfs filing system the filenames will need to use the standard short filename format.

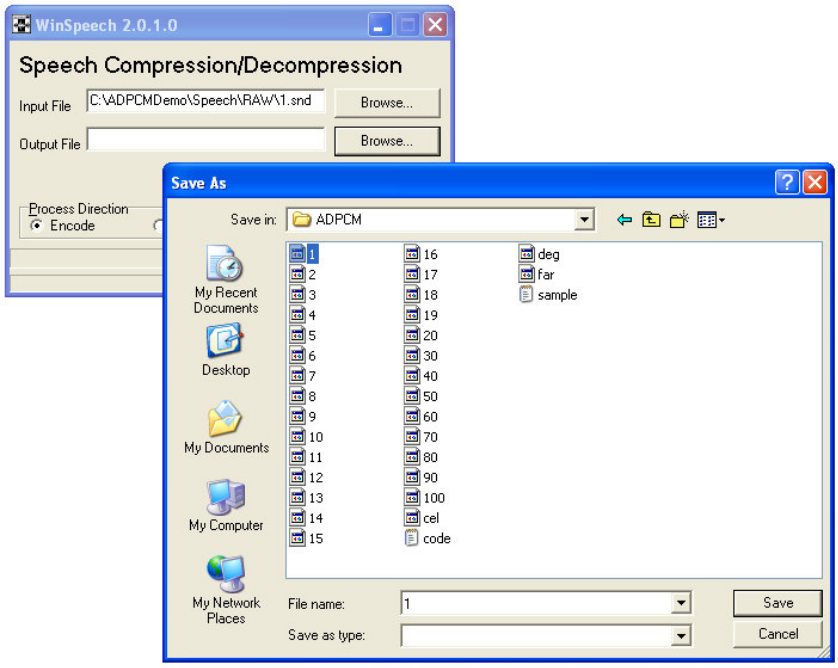

Fig 4: Browse for output file

Once the input and output files have been chosen, select the process direction – encode or decode. In this case encode is required, then press generate to create the compressed file.

If more than one conversion is required repeat the process for all files. At the time of writing, Winspeech cannot perform batch operations.

What is MPFS?

MPFS is a lightweight filing system for use exclusively on Microchip PIC microcontrollers and dsPIC® Digital Signal Controllers. It uses the standard 8.3 short file naming convention i.e a maxiumum of 8 characters for the filename with 3 characters for the file type extension. MPFS consists of a PC-based utility to create the image file plus the target source code driver required to access the images.

At the time of writing, MPFS does not support long filename access in the target device source code driver. Image files, when created on the PC, can use long filenames as they are used only as part of the build process within MPLAB. Short filenames are necessary for files added within the image file.

MPFS was created to provide a very small footprint file system which would, originally, allow the Microchip TCP/IP stack to follow standard web conventions which use short filename format to access data from web pages. We will be discussing the use of MPFS as related to speech samples only, since the utility has other features relevant to creating web page images which are beyond the scope of this document.

The MPFS file system utilises a small code footprint in the target device and also allows files to be converted into a format for storage in Program Memory or External Data EEprom devices using the MPFS utility.

The original version of MPFS was created for the Microchip PIC18 family of devices and related compilers. Therefore files targetted at the range of Microchip 16bit microcontrollers required some supplementary modifications by hand to be suitable for use with the C30 compiler.

To remove this obstacle, the MPFSv2 utility has been created. At the time of writing, MPFSv2 targets only the Microchip C30 compiler and therefore only 16bit devices from Microchip. It will also only create code suitable for storage in internal program memory. Future releases should see the scope of its capabilities increased. To create files compatible with MPLAB C18, the original command-line based MPFS utility should be used.

What is created?

MPFSv2 creates a single image file in MPLAB ASM30 assembler of type <filename>.s.

Once created <filename>.s file is added to the MPLAB project.

The MPFS image consists of a basic FAT table which references a filename to a memory location.

```
.byte   0x00,0x00
.long   paddr(_MPFS_0000)
.byte   '1','.','D','A','T', 0 , 0 , 0 , 0 , 0 , 0 , 0
```

The memory location itself is assigned a tag which is simply an assembler label. This allows the files to be easily referenced to a program memory location. A data table is created starting at the table entry label, _MPFS_nnnn, of size equal to the file being ported to MPFS, plus a minimal overhead for table handling.

```
;*********************************************************************************
; Original Filename: 1.dat
; MPFS 8.3 Filename: 1.DAT
;*********************************************************************************
        goto    END_OF_MPFS_0000    ; Prevent accidental execution of constant data.
        .global _MPFS_0000
_MPFS_0000:
        .pbyte  0xFC,0x10……
        .pbyte  ……
```

```
.pbyte    0x04,0xFF,0xFF,0xFF,0xFF          ; MPFS_ETX, MPFS_INVALID
END_OF_MPFS_0000:
```

Fig 5: Example of MPFS File System Entry

This simple file structure and handling method provides a compact method of handling files, which efficiently uses the table pointer mechanism in the 16-bit architecture to make best use of the 24-bit program memory word. This affords a more compact and efficient use of program memory relative to the use of the Program Space Visibility (PSV) feature of the Microchip 16-bit architecture, which would have used only 16 of the 24 bits.

Using MPFSv2

MPFSv2 can perform batch operations. This allows it to read all the contents of the files specified in the 'files to inlclude' pane and creates a series of entries in a single file, which can then be accessed by their short filenames using the MPFS source code.

Operation of MPFSv2 is relatively straightforward. Again, we advise that files are kept in a relevant folder within the project, especially if a number of files are to be added to the mpfs image, which is normally the case for speech and web based usage.

To locate the input folder, choose the relevant drive from the drop-down list. You should then be presented with a directory tree for the chosen drive. Locate the required input folder location from within the tree. A double-click on a folder will take you to the selected folder and allow selection of files.

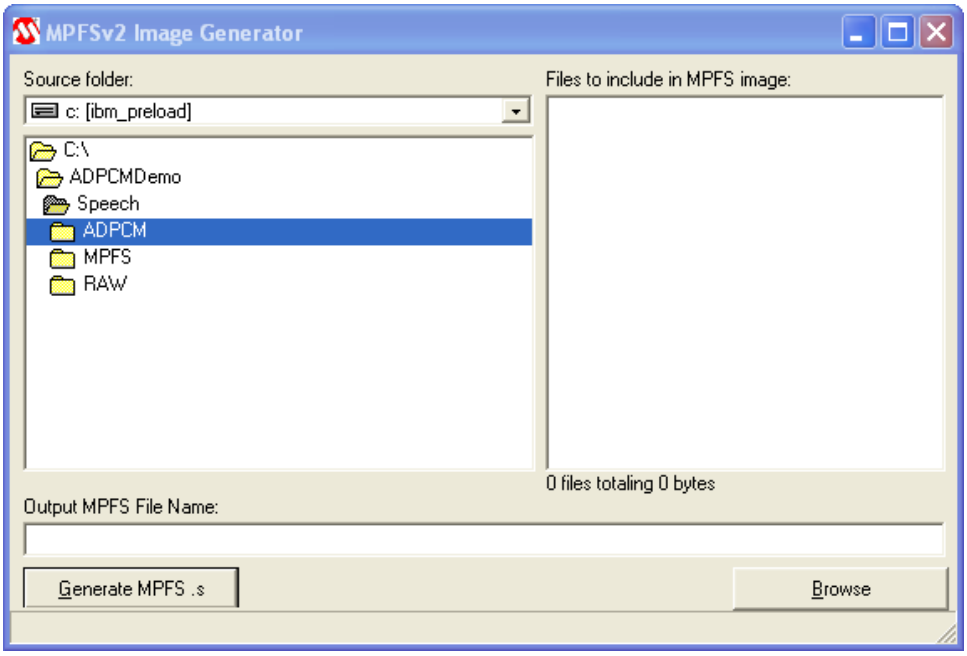

Fig 6: Locating the source folder

Once located, the available files will be presented in the right hand pane. Select all of the files which are to be added to the mpfs image.

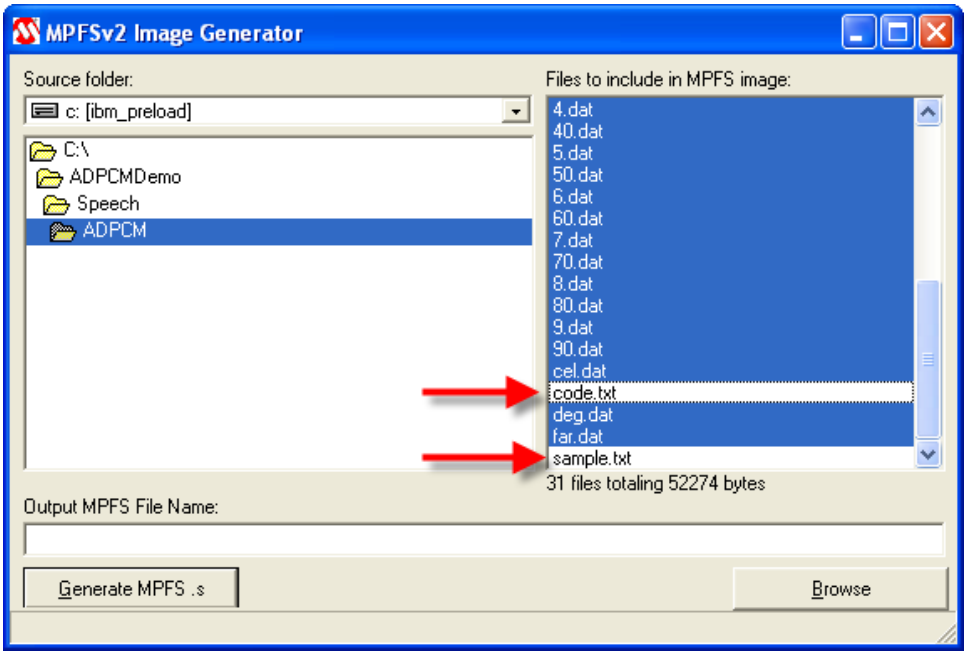

Fig 7: Selecting files in the source folder

Winspeech also creates two .txt files. These are not required in the mpfs image file and, if added, will use memory in the device. Therefore, select only those files with a .dat extension or any other files required specifically for the application.

The destination folder can be browsed using the browse button below the output filename. Once the destination folder is selected, a suitable filename should also be chosen.

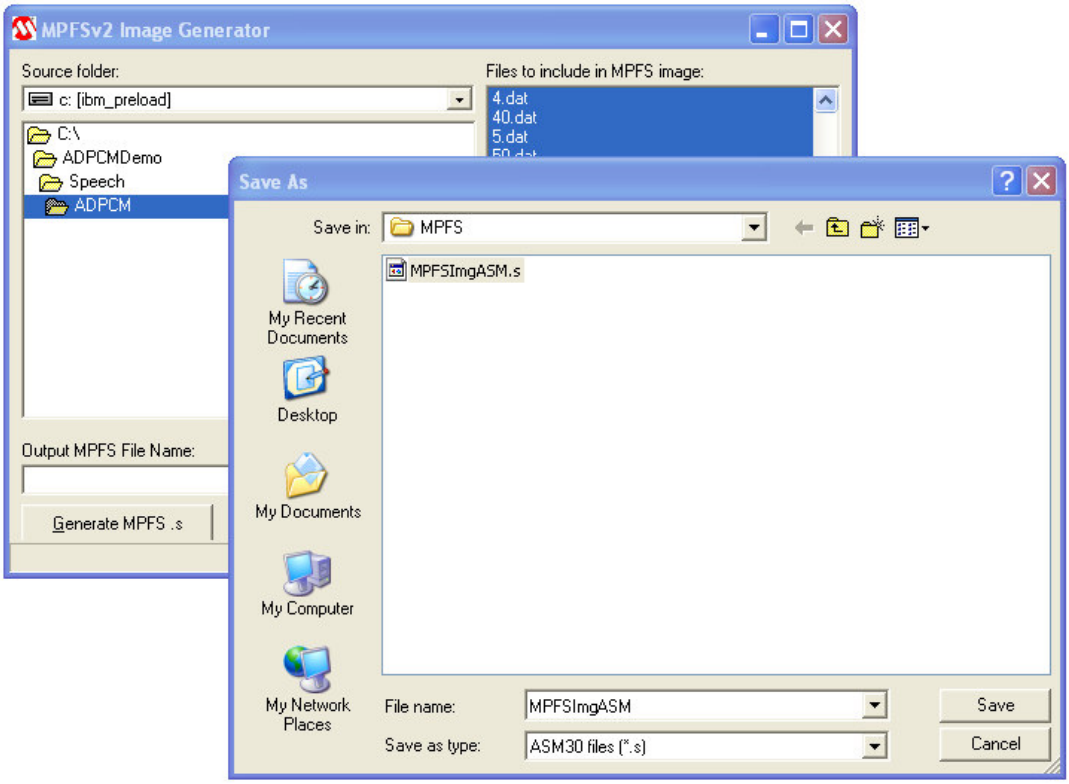

Fig 8: Destination browse and filename selection

In the case of the supplied example code, the required filename is MPFSImgASM.s.

Once the relevant folders and files are selected press the 'Generate MPFS .s' button.

If your target folder was a location other than your MPLAB project source location, you will need to copy the file to the source files location in your project and then add the MPFSImgASM.s file to your project file tree in MPLAB.

Simulation

Once we have created our sound files and manipulated them to create our ASM30 source file, we are presented with two options for simulating the project. Before either can be performed, we must ensure that we have added the MPFS image assembler code source file to the project file tree and then build the project. If build is successful, we can move ahead with simulation.

MPLAB

Simulation in MPLAB will allow us to perform a standard code simulation and remove some of the more obvious code errors. However, if we are to really determine if the code will function in a system-level environment, without hardware and debugging tools available, we need to make use of a system-level simulator such as Proteus VSM.

Proteus VSM

Proteus VSM is a simulation package from Labcenter Electronics, which will allow us to perform a full system-level mixed-signal simulation from within the MPLAB environment using the MPLAB plug-in capability. The implementation provided is a demo only and is minus a number of devices which would be needed to create a real Explorer-16 from the schematic, this is a powerful feature where we may simply wish to mock up the key components of a design. The Explorer-16 Virtual Demo Board does however contain the key sections required for our simulation namely the PIC24F device model, TC1047 temperature sensor and an implementation of the Speech PICtail Plus. The ability add virtual instruments such as a DVM,

oscilloscope or protocol analysers is also possible.The benefits of this include the ability, in our example case, to perform such tasks as:

– Change the temperature, through operation of the interactive buttons on the temperature sensor model, and observe related Vout with the virtual DVM

– Perform debug on our code to determine the ADC readings for a given temperature value and then manually cross-refer these to the values expected from the sensor data sheet

- Confirm that the ADC creates the correct conversion value for the temperature and voltage output relative to our chosen ADC reference voltage. Where necessary, we can iteratively fine-tune algorithms and modify sample and conversion times etc to suit.

– Once we are sure that the correct input operations are being performed, we can perform a batch process to obtain the complete speech output over time and then play this output back on the PC. This is one of the mixed-mode microcontroller and spice simulation capabilities that Proteus can perform to provide system-level debug.

– We can modify the output filter stage to change its characteristics and then re-test. All of which can be performed in a fast, easy iterative manner without any hardware.

Finally, we can try it out on real hardware to confirm our simulation. The versatility and features available from a tool such as Proteus VSM allow us to model our design in a much shorter time frame, analyse and solve debug issues from a system perspective and perform a controlled iterative set of tests. This is performed without a commitment to physical hardware before it is either ready or available. This capability can save a significant amount of time and effort on the bench and should help us to reduce the design cycle time.

With the combined capabilites of MPLAB and Proteus VSM, we now a have a way of performing system-level debug and simulation quickly and easily. Of course, the fact that VSM uses the same simualtion operations as MPLAB means that moving to a hardware debug tool to try the same operations is also a simple intuitive operation.

Hardware

Once we have performed the system level simulation, we are ready to commit to hardware. This may, of course, require an element of debugging the hardware itself. However this process should be eased because we have performed sufficient system level simulation on our code to have a higher degree of confidence that it actually works. It should also help us reduce bench-test time, where we chase around in a loop fixing hardware and software faults where we were only able to test some operations in a standard ‘code simulation only’ environment.

Explorer-16

For this development, we have targeted a known piece of hardware in the Microchip Explorer-16 development board. Therefore, moving between our simulation and hardware should be as easy as programming the device and observing the same results in hardware as we do on the simulator.

Audio PICtail Plus

To help ease our development cycle and make use of readily available, low cost, hardware demonstration kits, the Explorer-16 is equipped with an expansion connector known as the PICtail Plus interface. This allows us to place a number of hardware modules onto the Explorer-16 board to help conceptualise our needs in hardware before commitment to a final design.

In this instance we are using the Audio/Speech PICtail plus. This provides us the ability to output our speech samples in real time using either the on-board mini speaker or connect to an external speaker or headphones using the jack socket.

The board itself provides a simple volume control and 4^{th} order filter stage to remove some of the unwanted components introduced during sound reproduction.

Programmer

To program our PIC24F device onboard the Explorer-16 we have a number of options, some of which are outlined here.

PICkit™ 2

PICkit 2 is a low cost programmer that provides a means to perform programming-only operations on the PIC24F device. The full capabilities of the programmer can be found in the relevant documentation. The PICkit 2 programmer uses a stand alone interface, separate to MPLAB, which requires that you locate the .hex file for the project and open this with the PICkit 2 interface. The essential operations relative to PIC24F programming: Program, Read, Verify, Erase, can then be performed.

PICkit 2 should be sufficient for most cases where we have managed to remove the bugs from our code in the simulator and are using the hardware for confirmation of operation in the real world.

MPLAB ICD2

Where you find that a bug simply doesn't show up in the simulator for whatever reason then a tool capable of both programming and debug operations, such as ICD2, may be required. This will allow us to compare simulation with the hardware, using breakpoints in relevant places and looking at physical states of the hardware. This ability to compare expected with actual may prove to be a bi-directional debug, but it should help us to remove complex issues from our designs significantly quicker than through basic code-only simulation and hardware debug alone.

MPLAB REAL-ICE

A more recent addition to the Microchip programming and debug arsenal is the MPLAB REAL ICE™ In-Circuit Emulator. This extends the capabilites of ICD2 through provision of port trace capability, instruction capture trace and logging, real time watch, stopwatch and other features. Connection to the PC is via a high-speed USB2.0 interface with options for device interconnect of the standard RJ11 type connector or a high-speed, noise-tolerant LVDS interface pack.

Limitations and requirements

PIC24

Available memory and target compression algorithm relative to device capabilities ultimately limit the quantity of speech/audio that we can play back.

The method we are using to store sound uses the program memory of the target device. This has many benefits in a number of systems where only short sound bites are required or, as we have done, messages can be created using individual words to generate larger messages from a limited vocabulary.

In our case, we have used an ADPCM compression algorithm to reduce the memory footprint further. This, combined with the efficent use of memory space afforded by the mpfs file system, allows more or larger sound bites to be stored. Since we are using a generic microcontroller, we are also using a simple compression algorithm rather than a more complex option which might require dsp capability, a significantly faster processor or dedicated hardware. An example of this would be Speex, which requires the capabilites of the Microchip dsPIC Digital Signal Controller to provide the additional maths acceleration required to decompress data in a short time relative to our application. The dsPIC DSC also offers additional capabilities to allow algorithms such as G.711 and G.726 to be used via additional libraries. G.711 is a simpler 2:1 compression ratio ADPCM algorithm and will therefore run on a PIC24 device also, however, the sample rate of the algorithm is quite high so the knock on effect on memory, as discussed above, will occur.

The table below shows a number of available compression methods and the devices supported

	dsPIC30F	dsPIC33F	PIC24H	PIC24F
SPEEX Speech Encoding/Decoding (SW300070)	☑	☑	☒	☒
G.711 Speech Coding/Decoding (SW300026)	☑	☑	☑	☑
G.726A Speech Coding/Decoding (SW300090)	☑	☑	☒	☒
IMA ADPCM Speech Coding/Decoding (DS00643B – AN643)	☑	☑	☑	☑

Fig 9: Available compression algorithms and devices supported

Storing in internal program memory allows efficient use of a device, reduces the component count and keeps costs under control. Where larger sound bytes, bigger vocabulary or higher sample rate source data are required, it is possible to add external storage such as DataEE or Serial Flash or alternatively use mass storage devices such as MMC/SD/CF cards. This is an option we shall look at in another article.

Another trade-off to consider is the effect the desired frequency of the output has on the system. Since we are using a PWM-based DAC and the PWM period is a function of the system clock frequency, higher resolution and higher frequency output will require higher system clock speeds. Higher speeds require more power, so some system level decisions will need to be made, especially if considering a battery powered or low-current application.

Consideration needs to be given to the perceived quality of the audio output, taking into account the quality of the final output stage. While 8kHz is good enough for audible reproduction, is it acceptable to the end user ? If not, what is ? A compromise between specmanship and actual audio quality has to be made – in most cases we are playing back on simple low-cost systems that will affect the output quality more than we gain by increasing the sample rate and resolution.

The flexible nature of the design approach we have taken here enables us to modify the output period quickly to provide say 12kHz, 16kHz or any of the myriad other choices of output at a higher bit resolution. We can quickly re-sample the files to provide the correct input and allow us to test and compare the differences to work out how much of a difference the changes make. We can then make an evaluative decision which includes cost factors, power budgets etc. We can also help to determine if changes in the analogue sub-system are more suitable for our design. The system simulation approach again allows us to make rapid changes to our design and determine where to make the first pass hardware.

We may also determine that an increase in sample rate or resolution has a knock-on effect on device clock speed. We can manage the PWM side of this to some extent through the use of a dual output-PWM, which is externally weighted and summed. This may also allow us to reduce our clock frequency and hence improve our power budget for a desired output sample rate. However, we will always need to ensure that we have sufficient processing time available to decompress the ADPCM samples at the chosen clock frequency as we perform conversions on the fly.

TC1047

The TC1047 is a voltage output temperature sensor. The data sheet shows that it is accurate to +/-2 degC max at 25 degC and +/-3 degC max at 125 degC.

Given the accuracy of the temperature sensor, it is likely that we will see some deviation between actual temperature and the temperature spoken. A number of other factors may also affect this, including stability of the voltage reference to the TC1047 and ADC, localised heating, noise at time of ADC reading, board layout etc. It is also likely that a system-level decision needs to be taken as to how much time is spent refining the algorithm used to derive the temperature from the ADC reading to cope with these changes. Ultimately some error will also creep into the system through rounding and maths errors, so again a system-level design choice has to be made on how to best minimise this and ensure that the design remains within the accuracy of the sensor. This has to be done without spending too much time on the algorithm and testing at design time, or creating a test method that requires excessive time and setup during manufacture.

The model for the TC1047 accurately follows the typical curve in the datasheet, this can be checked by running the simulation, modifying the TC1047 output value and viewing this on the DVM.

The algorithm used in the code also follows the data sheet recommended calculation. However, we are using integer maths, so the rounding operations affect the basic calculation and as a result the displayed temperature on the TC1047 and spoken temperature may differ. It may therefore be possible to refine the algorithm and compare the expected reading against the actual to determine if an error exists in the reading, calculation or elsewhere. In our case we were able to determine a rounding error through iterative analysis of the results and therefore modify the algorithm to add a weighting to help offset the rounding and increase the accuracy of the system. With this weighting factor, the error appears to be within the limits defined for the temperature sensor. However, within an end product, further testing or refinement may be required.

Conclusion

The details provided should give the reader sufficient information to create sound samples and perform a playback operation on the PIC24F microcontroller. The same techniques can be applied to other PIC microcontrollers with allowance being made for device performance and availability of suitable tools for file system construction. While the application shown was for a speaking thermometer, the same methods can be used to create other applications which would benefit from a voice-based feedback mechanism, with the provided examples serving as a template. Certainly the Virtual Speech/Audio PICtail demo will allow access to a rapid evaluation method prior to creating hardware targeted at any given application.

Appendix

A.1. Flow Diagram

A flow diagram showing the process of creating, manipulating and storing speech samples for playback is shown below.

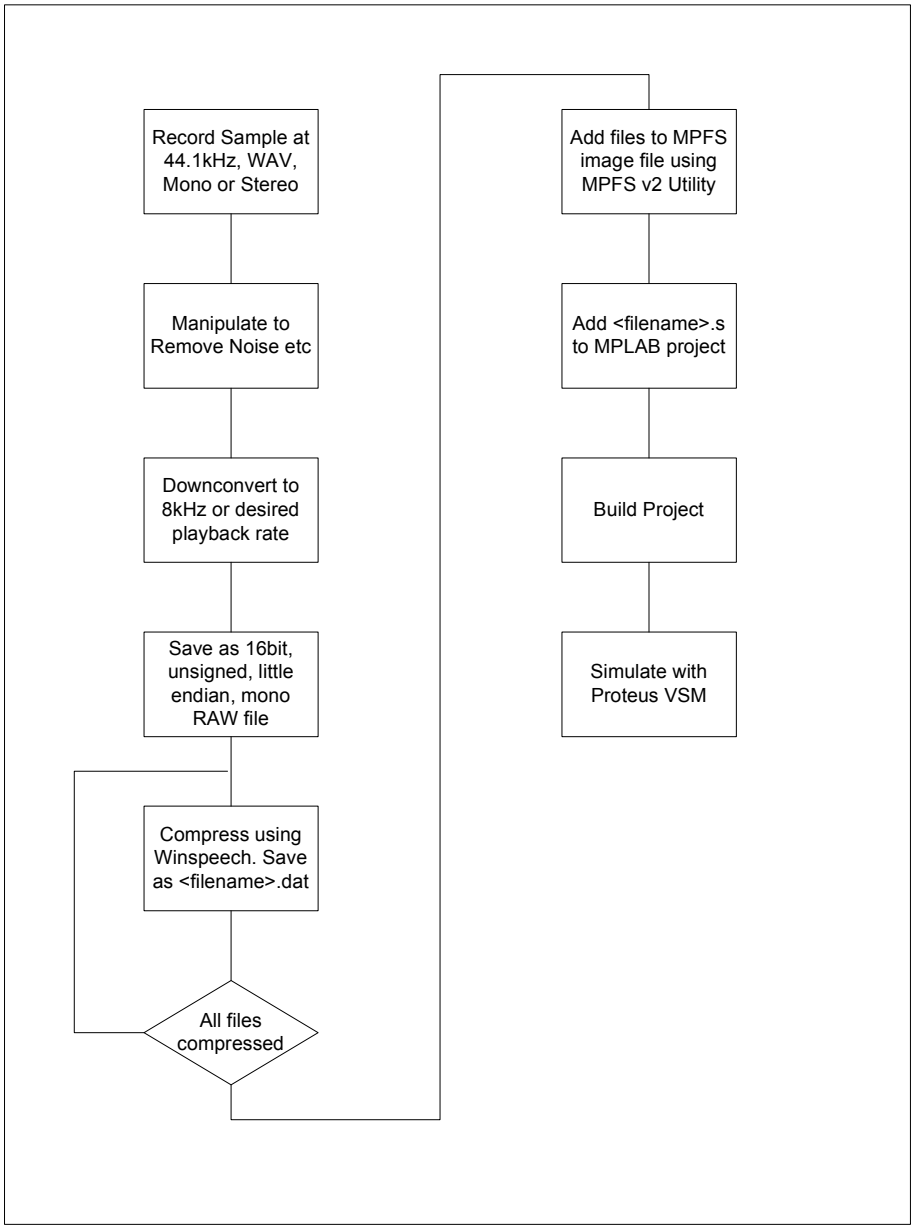

Fig.A.1. Speech sample and playback process flow

A.2. Sound File Formats

The sound file formats for each of the stages described are shown here.

	Original Recording	Post Manipulation	Compressed
Sample Rate	44.1kHz	8kHz*	8kHz*
Stero/Mono	Stereo or Mono	Mono	Mono
File Format	.WAV, PCM Signed 16bit	.RAW, .SND, PCM Unsigned 16bit little endian	.DAT, compressed ADPCM

* The rate to which the speech is downsampled or converted can be other than 8kHz. However, the PWM rate would need to be modified to suit and allowance made for sample size and available memory.

A.3. Audio/wave editing tools

A number of wave editing tools are available which may be suited to the manipulation requirements outlined. This list is by no means exhaustive and only a small number have been tested for suitability. Some of the available tools are shown below,

Goldwave - ***http://www.goldwave.com/***

Wavepad - ***http://www.nch.com.au/wavepad/masters.html***

Audacity - ***http://audacity.sourceforge.net/***

Fleximusic - ***http://www.fleximusic.com/***

Acoustica - ***http://www.acoustica.com/***

* Remember however that the required output for processing via Winspeech must be met i.e. PCM Unsigned 16bit Little Endian.

A.4. Proteus

A demo version of Proteus for PIC microcontrollers can be downloaded from the development tool downloads area on www.microchip.com. Alternatively please contact Labcenter Electronics direct for more information and details

http://www.labcenter.co.uk

A.5. Basic code operation

A demo for the speaking thermometer outlined in the article can be downloaded from, http://www.elektor-electronics.co.uk/Default.aspx?tabid=140 (Demo4.zip, April 2007 issue). This provides the example code, Explorer-16 Virtual Demo Board, MPFS and Winspeech utilities. To help understand the source code operation some details are provided below.

Main Loop

Following initialisation, the code will sit in a loop waiting for button S4 on the Explorer-16 to be pressed. If batch mode analysis is being performed in Proteus VSM then a stimulus injector is included on the S4 net which simulates a button press and allows the batch mode simulation to run.

ADC Read

Once S4 has been pressed, a read of the ADC is initiated. This triggers an ADC GO operation after which the code waits until the ADC DONE bit is asserted. This is potentially less efficient than using interrupts. However, we are creating a single stimulus and no other events are operating in parallel.

Once the ADC reading is obtained, it is normalised to create the temperature sensor output value in degrees C relative to the reading. An additional weighting is added to the calculation aid rounding and integer maths errors.

Once determined, the temperature value is sent to the decode routine which determines which soundbites need to be played to speak the temperature.

MPFS

A filename is created for each section of the playback phrase required. In this case the filename is used to access a file using the MPFS file system but the same filename can also be used to initiate file access using the FAT16 filing system.

PlayClip

The Playclip function is called for each filename generated. This function extracts the data from the file and passes it to the ADPCM decode routine. Once decoded, the function also handles sending data to the output compare module for generation of the PWM output. The output compare is set up to operate at 16kHz with a 10bit output. This can be modified for a different speed or sample size. In this instance 8kHz is achieved by generating a second interrupt with the same OC output value loaded. A resolution of 10bits is chosen as this matches the ADC resolution of the PIC24F were the system able to record speech. It also helps to reduce the additional clock speed that would be needed for increased resolution at the same 8kHz or 16kHz output frequency.

Once a complete message converted temperature has been decoded, the code returns to the loop waiting for S4 to be pressed.

A.6. Further Excercises

– Create the sound file in a local language. Manipulate to create individual sound clips and follow the process flow to create your own MPFSImgASM.s file

Probably the easiest way of creating the individual sound bites required is to generate one .wav file with all the required components. The relevant sample can then be extracted using the wave editor of choice and the re-sampling and edit operations performed.

Performing operations such as noise and hiss removal, equalisation, filtering etc on the complete set of samples is simpler if they are all in a single file. This ensures that the same changes are made to all files in the same manner. All of these operations should be performed prior to extracting individual sound bites.

To create the basic sound file you will need to record the following set of words. Orate them clearly and leave a suitable gap between each to allow you to edit them easily in the wave editor.

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,30,40,50,60,70,80,90,100, Degrees, Celcius, Fahrenheit

Where an individual language requires you may also need to add a additional word to the above list. An example of this would be German, where we would require:

Vier und vierzig	for 44 as opposed to
Forty four	as currently defined in English.

We therefore need to record ‘und’ as an additional sample and ensure that we construct the output correctly in the source code to suit the local language. The exact set of numbers required will of course differ between languages, as will the changes to the code to manage some of the differences in pronunciation.

– When successfull create additional sound samples for

zero, minus, two hundred

Edit these as required and then create a complete new MPFSImgASM.s file containing all the samples.

– Add code required to use full working range of TC1047 temperature sensor -40 to +125degC

The code, as supplied, does not have speech samples for minus and zero. As a result it only provides speech output for +1degC to +125degC. If we wish to output negative temperature we will need to add the zero and minus words to the vocabulary.

– Add code to output temperature in Fahrenheit

If we wish to use Fahrenheit as our output temperature, we will need to modify the conversion algorithm to generate the correct value.

References

DS00643B – AN643 Adaptive Differential Pulse Coded Modulation using PICmicro Microcontrollers

Richey, Rodger

Microchip Technology Inc. 1997

Implementing Speech on 8bit Microcontrollers

Richey, Rodger

Embedded Systems Magazine pp 31 – 40, April 2000