1.0 INTRODUCTION

Functions are the basic building blocks of the C programming language. Use of functions allow various tasks to be performed without the need to have a detailed knowledge of how this is achieved. The 834KEIL.lib is a library of functions, currently targeted at the Keil environment, allowing easier C programming of the ADuC834. Most of the functions have a hardware independent interface, which means that there is no need to know which register to program or how to program it.

This technical note

- Shows by way of example the benefits of using the ADuC834 C library
- Describes how to use the C-library provided for the ADuC834
- Lists the functions implemented
- Gives a complete function description for each function.

1.1 EXAMPLE SHOWING BENEFITS OF USING THE ADUC834 C LIBRARY

Below is a small example of C code that has been written using the C functions provided. This simple program performs ADC conversions in continuous mode. If a valid reference is found then the ADC result will be sent up the UART serial port to the PC at 9600 baud. If there was an error with the reference then an error message is sent up the UART instead.

From the example below it is clear that the use of functions has made the C code much simpler to write and to read.

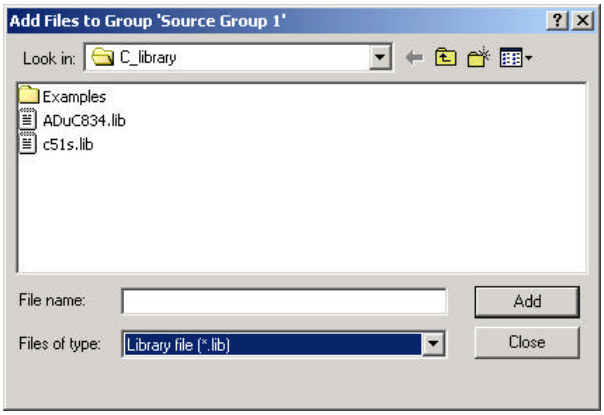
```
#include<ADuC834.h>                     // SFR definition file.
#include"lib834.h"                      // Function and variable declaration file.
#include<stdio.h>
#include<stdlib.h>

void main (void)
{
    char cErr;
    PllWcd(1);                          // PllWcd writes CD bits. core frequency at 6.291456MHz
    UrtCfg(0x12,-20);                   // Configure UART at 9600 baud using Timer2

    AdcCfg(0x60,0x00,0x0D);          // Primary ADC channel AIN2/2
                                        // external ref
                                        // bipolar mode
                                        // +/- 20mV input range
                                        // conversion rate at 5Hz

    AdcGo(3);                           // Start continuous conversions

    while(1)
    {
        while(((cErr = AdcErr())&0x80));        // Check if conversion on primary ADC is finished
        if (cErr&0x10)                          // Check if reference error
        {
            printf("reference error %\n");
            while(((cErr = AdcErr())&0x10));    // Wait for correct of reference
        }
        else printf("result of conversion on primary ADC: %ld\n\n",AdcRd(0));
    }
}
```

1.2 HOW TO USE THE C LIBRARY

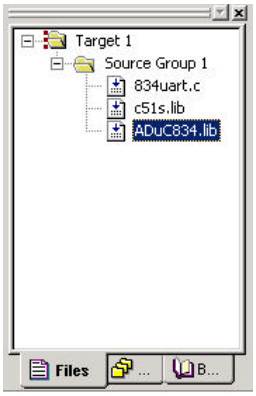
The C library can be used very easily from the Keil µVision2 environment. Three files are required to use the library; the library itself (834keil.lib), the header file to declare the functions and variables of the library (lib834.h) and a modified C51s.lib library file to increase the functionality of the putchar() and _getkey() functions.

The **834keil.lib** file is the object code of the functions generated by the Keil compiler. This file should be included in the project with the source file(s). This is easily done as follows:

1. From the Project dialog box, click on Add to add a new source file.
2. From the file types list, select Library files (*.LIB).
3. Select the library file using the File to Add dialog box as shown below.
4. Click on ADD.



µVision2 will automatically add the library file to your project. The project window should get updated as follows.
(Note. This picture also shows the inclusion of the C51s.lib library as discussed later)



As shown in the example above the **lib834.h** file must be included with the source file. If lib834.h is located at C:\ADuC\C-Lib\834\ then the following line must be included at the start of your code.

```
#include "C:\ADuC\C-Lib\834\lib834.h"
```

Keil includes a **C51s.lib** library file at C:\Keil\C51\lib\c51s.lib. This file includes the _getkey() and putchar() functions. Optimized versions of these functions (written for the ADuC834) exist in our version of C51s.lib. To use these optimized versions over the Keil versions then the c51s.lib file must also be included in your project. This is done in the same way as discussed for the 834keil.lib file earlier. The Keil version of C51s.lib does not have to be overwritten.

2.0 LIST OF FUNCTIONS

The 834keil.lib contains the following list of C functions.

PERIPHERAL	FUNCTION	SHORT DESCRIPTION
ADC	AdcCfg()	Configure ADCs
	AdcGo()	Start conversion(s)
	AdcErr()	Get ADCs status (and errors)
	AdcRd()	Read conversion result
	AdcGet()	Get ADC conversion
DAC	DacCfg()	Configure DAC
	DacOut()	Output a value on DAC
PLL	PllDly()	Software delay
	PllFst()	Select interrupt speed
	PllWcd()	Write CD bits
	PllRcd()	Read CD bits
TIC	TicGo()	Set and start timer
	TicHr()	Reads HOUR
	TicMin()	Reads MIN
	TicSec()	Reads SEC
	TicHth()	Reads HTHSEC
	TicVal()	Set and start interval counter
UART	UrtCfg ()	Configure UART and baud rate
	UrtBsy ()	Checks UART busy status
	Putchar	Sends character via UART
	_getkey	Gets character received by UART

Table 1: list of functions in 834keil.lib

By including the lib834.h file, memory space is also allocated for the two following variables, which are used by the functions:

cUrtVar Status of UART
pcAdcBuf[6] Buffer with result of conversion.

2.1 DESCRIPTION OF THE ADUC834 C-LIBRARY FUNCTIONS

AdcCfg

Function prototype: char AdcCfg (char cAd0, char cAd1, char cSf);

Description of Function: The AdcCfg function configures the primary and auxiliary ADC for reference and channel selection, unipolar or bipolar coding and in the case of the primary ADC for range. It also sets the sinc filter register SF that controls the update rate.

User interface: Into cAd0 put desired ADC0CON.
Into cAd1 put desired ADC1CON.
Into cSf put desired SF.

The AdcCfg function configures the ADC by writing to the ADC0CON, ADC1CON, and SF registers. This function does not start ADC conversions and hence the ADCs are left in power down mode. If configured using an external reference, the function checks to see if the reference is present or not.

Return value: The AdcCfg function returns 0 if the external reference is not present but required and 1 if the external reference is present or not required.

Example: The following example configures the primary ADC, channel on AIN1/2, with internal reference, in bipolar mode, with +/- 640mV input range. It does not change the configuration of the Aux ADC and it also leaves the update rate (SF) as it is.
AdcCfg (0x45, ADC1CON, SF);

Side effects: Overwrites ACC and P.

AdcGo

Function prototype: char AdcGo(char cMode);

Description of Function: The AdcGo function initiates conversion on primary and auxiliary ADC.

User interface: Set up the ADC using AdcCfg.
Put required mode bits MD0 to MD2 in cMode.
0 corresponds to the ADC Power Down Mode,
1 corresponds to Idle Mode,
2 corresponds to Single Conversion Mode,
3 corresponds to Continuous Conversion Mode,
4 corresponds to Internal Zero-Scale Calibration,
5 corresponds to Internal Full-Scale Calibration,
6 corresponds to System Zero-Scale Calibration,
7 corresponds to System Full-Scale Calibration.
Calling AdcGo starts conversion according to mode bits.

Return value: The AdcGo function returns cMode.

Example: The following example starts a conversion in Continuous Conversion Mode:
AdcGo(3);

Robustness: Conversion can take a significant time depending on SF. The AdcErr function can be used to check for completion.

Side effects: Overwrites ACC, CY, P.

AdcErr

Function prototype: char AdcErr(void);

Description of Function: The AdcErr function returns the status of the ADC. If the ADC(s) is/are finished converting, the function puts the ADC results into pcAdcBuf. This buffer will not be overwritten unless you call AdcErr or AdcGet again.

User interface: When a result is ready:
Result of the primary ADC is put in pcAdcBuf[0to2].
Range bits are put in pcAdcBuf[3] bits 0 to 2.
Result of the auxiliary is put in pcAdcBuf[4to5].

Return value: Zero indicates valid result in pcAdcBuf.
Bit 0 set indicates Aux ADC powered down or idle.
Bit 1 set indicates Primary ADC powered down or idle.
Bit 2 set indicates Aux ADC out of range error.
Bit 3 set indicates Primary ADC out of range error.
Bit 4 set indicates reference error.
Bit 5 set indicates last conversion was calibration.
Bit 6 set indicates Aux ADC result not ready.
Bit 7 set indicates Primary ADC result not ready.

Example: The following example waits until primary and auxiliary ADC results are ready:
```
while(AdcErr()&0xC0);
```

Side effects: Overwrites ACC, P and 6 bytes in DATA memory at pcAdcBuf.

AdcRd

Function prototype: long AdcRd(char cAdc);

Description of Function: The AdcRd function reads Primary or Auxiliary conversion result from pcAdcBuf.

User interface: To return the primary ADC result set the parameter to 0.
To return the auxiliary ADC result set the parameter to 1.

Return value: If cAdc==0 the AdcRd function returns result from the primary ADC, else from the auxiliary ADC.
The return value is 32 bit signed long with
7FFFFFFF corresponds to +2.56V
00000000 corresponds to 0V
80000000 corresponds to -2.56V

Example: To display the result of the conversion on primary ADC:
```
printf("Primary ADC conversion result in decimal: %ld\n",AdcRd(0));
```

Robustness: In unipolar mode the return value is still signed but negative values are not possible.
To ensure values are valid the AdcErr function must be used.
The result is independent of the range bits if the signal is in range, but optimum range bits will give best accuracy.

Side effects: Overwrites ACC, P, CY, R0.

AdcGet

Function prototype:	long AdcGet(int* piVal);
Description of Function:	The AdcGet function requests a conversion and returns with result when available.
User interface:	Put address of piVal in input parameter and call AdcGet. AdcGet does a quick conversion to select the best primary ADC range. AdcGet then executes AdcGo (performing a single conversion) followed by executing AdcErr repeatedly until a conversion result is available. AdcGet returns the result of the primary ADC using AdcRd while the result of the auxiliary ADC is put at piVal.
Return value:	The function AdcGet returns the result of the primary ADC.

Example:	int iPrim; //primary ADC raw result using AdcGet();
	AdcGet(&iPrim);

Robustness:	AdcErr cannot be used to get ADC status as the ADC is powered down at the end of AdcGet.
Side effects:	Overwrites ACC, P, R0, R1 and 6 bytes in DATA memory at pcAdcBuf. Uses 3 stack levels.

DacCfg

Function prototype:	char DacCfg(char cDaccon);
Description of Function:	The DacCfg function puts parameter cDaccon in the DACCON SFR.
User interface:	Set the following bits of cDaccon: Bit 0 to power up DAC. Bit 1 not to zero DAC. Bit 2 for AVdd full scale (else Vref). Bit 4 for Output on AIN4 (else AIN2). The DacCfg function writes cDaccon in DACCON.
Return value:	The DacCfg function returns cDaccon.
Example:	The following example configures the DAC output to be on P1.7, with a range of 0 – 2.5V in normal DAC operation and powers on the DAC: DacCfg(0x16);
Side effects:	Overwrites ACC, P.

DacOut

Function prototype:	char DacOut(int iDac);
Description of Function:	The DacOut function causes iDac to be output on DAC pin.
User interface:	Set DAC options by executing DacCfg, call DacOut. iDac is output on DAC pin.
Return value:	The DacOut function returns 1 if iDac is in range, else 0.
Robustness:	If iDac>4095 output full scale and if iDac<0 output 0.
Example:	This example outputs 1.33V based on 2.5V full-scale range: DacOut(3000);
Side effects:	Overwrites ACC, P.

PllDly

Function prototype:	char PllDly(uint uDlyMs);
Description of Function:	The PllDly function causes software delay of uDlyMs milliseconds.
User interface:	Put delay in uDlyMs.
Return Value:	The PllDly function always returns with 1.
Robustness:	Due to unavoidable rounding the delay times have typical errors of up to 3%. An offset of about $15\ \mu s \times 2^{CD}$ becomes dominant for large CD and small delays. Delay will be increased if normal interrupts occur during delay although the delay can be decreased if short "fast interrupts" occur during delay.
Example:	To include a 10ms delay: PllDly(10);
Side effects:	Overwrites ACC, P and IEIP2.

PllFst

Function prototype:	char PllFst(char cEn);
Description of Function:	The PllFst function selects if interrupts will run at fast or normal speed.
User interface:	For normal interrupts write zero to cEn. For fast interrupts write non-zero to cEn.
Return value:	The PllFst function returns previous setting.
Robustness:	No known issues.
Example:	To enable fast interrupts PllFst(1);
Side effects:	Overwrites ACC, P and CY.

PllRcd

Function prototype:	char PllRcd(void);
Description of Function:	The PllRcd reads the CD bits.
Return value:	The PllRcd function returns CD bits.
Robustness:	No known issues.
Example:	To use the CD bits as a condition, the variable cCd contains the CD bits cCd = PllRcd();
Side effects:	Overwrites ACC and P.

PllWcd

Function prototype: char PllWcd(char cCd);

Description of Function: The PllWcd function writes CD bits, which determine the frequency at which the microcontroller core will operate.

User interface: Put CD value in cCd:
0 corresponds to a core frequency of 12.582912MHz,
1 corresponds to a core frequency of 6.291456MHz,
2 corresponds to a core frequency of 3.145728MHz,
3 corresponds to a core frequency of 1.572864MHz (default value),
4 corresponds to a core frequency of 0.786432MHz,
5 corresponds to a core frequency of 0.393216MHz,
6 corresponds to a core frequency of 0.196608MHz,
7 corresponds to a core frequency of 0.098304MHz.

Return value: The PllWcd function returns CD bits.

Example: To set the core frequency at 6.291456MHz:
PllWcd(1);

Robustness: Only 3 LSBs of parameter1 are used.

Side effects: Overwrites ACC and P.

TicGo

Function prototype: char TicGo(char cHr, char cMin, char cSec, char cHth);

Description of Function: The TicGo function sets hour, minute, second and hundredths and starts TIC.

User interface: Set the 4 parameters to the required time.
Call TicGo function.
TicGo sets the start time and starts the timer.

Return value: The TicGo function returns TIMECON (guaranteed not zero).

Example: To set 12h30:
TicGo(12,30,0,0);

TicHr

Function prototype: char TicHr(void);

Description of Function: The TicHr function reads HOUR counter.

Return value: The TicHr function returns HOUR.

Example: printf("Time now: %02bdh%02bdm%02bds,%02bd.\n",TicHr(),TicMin(),TicSec(),TicHth());

Side effects: None.

TicMin

Function prototype:	char TicMin(void);
Description of Function:	The TicMin function reads MIN counter.
Return value:	The TicMin function returns MIN.
Example:	printf("Time now: %02bdh%02bdm%02bds,%02bd.\n",TicHr(),TicMin(),TicSec(),TicHth());
Side effects:	None.

TicSec

Function prototype:	char TicSec(void);
Description of Function:	The TicSec function reads SEC counter.
Return value:	The TicSec function returns SEC.
Example:	printf("Time now: %02bdh%02bdm%02bds,%02bd.\n",TicHr(),TicMin(),TicSec(),TicHth());
Side effects:	None.

TicHth

Function prototype:	char TicHth(void);
Description of Function:	The TicHth function reads HTHSEC counter.
Return value:	The TicHth function returns HTHSEC.
Example:	printf("Time now: %02bdh%02bdm%02bds,%02bd.\n",TicHr(),TicMin(),TicSec(),TicHth());
Side effects:	None.

TicVal

Function prototype:	char TicVal(char cVal, char cMode);
Description of Function:	The TicVal function puts interval counter in operation.
User interface:	Set first parameter to required interval count and second parameter to choose the counting interval as follows: 0 to count in 1/128th of second (HTHSEC), 1 in seconds (SEC), 2 in minutes (MIN) and 3 in hours (HOUR) Add 8 to this parameter for a single interval (otherwise multiple intervals are measured). TicVal zeroes interval counter sets required interval count and clock source and starts interval counting. Enables global interrupt bit and TIC interrupt on low priority. When the required interval is reached, interrupt TicInt is executed.
Return value:	The TicVal function returns TIMECON (guaranteed not zero).
Example:	To enable Interrupt on 10 counts of SEC counter: TicVal(10,1);
Side effects:	Starts TIC timer if not running already.

UrtBsy

Function prototype: char UrtBsy(void);

Description of Function: The UrtBsy function checks the UART busy status.

User interface: One byte must be allocated in DATA segment for cUrtVar.
Call UrtBsy.

Return value: The UrtBsy function returns:
- 0 if ready.
- with bit 0 set if a byte received.
- with bit 1 set if a byte can be sent.
- with bit 7 set if an error occurred.

Example: To check if a character is received
if(UrtBsy()&1);

Side effects: Overwrites ACC, CY, P and R1.

UrtCfg

Function prototype: char UrtCfg(char cCfg, int iDiv);

Description of Function: The UrtCfg function configures UART and selects baud rate.

User interface: Make sure than the CD bits are correct first.
Into cCfg put:
0 to disable baud rate generation.
1 to use Timer 1.
2 to use Timer2.
3 to use fractional divider (Timer 3).
To cCfg add:
0 for no parity 1 stop bits.
4 for no parity 2 stop bits.
8 for odd parity 1 stop bit.
12 for even parity 1 stop bit.
Put the reload value required for baud rate in iDiv. If using timer 1 only the low byte of iDiv is used for the reload value. If using timer 3 the low byte of iDiv is T3FD and the high byte is T3CON.
cCfg is put in cUrtVar with MSB cleared.

Return value: The UrtCfg function returns cUrtVar.

Examples: for a baud rate of 9600, using Timer 1:
PllWcd(0);
UrtCfg(0x01,-7);
For a baud rate of 9600, using Timer 2:
PllWcd(1);
UrtCfg(0x12,-20);
For a baud rate of 9600, using Timer 3:
PllWcd(3);
UrtCfg(0x33,0x8512);

Robustness: CD bits must be correct when communicating.

Side effects: Overwrites ACC, CY, P and R1.
UART register SCON is changed.
If T1 is used TMOD, TCON, PCON.7 and TH1 are changed.
If T2 is used TMOD, T2CON, RACP2H and RCAP2L are changed.
If fractional divider is used T3FD and T3CON are changed.

putchar

Function prototype: char putchar(char cTx);

Description of Function: The putchar function sends a character via UART.

User interface: UART must be initialized with UrtCfg() before calling putchar. Put character to send in cTx and call putchar. putchar waits if UART busy then adds the required parity and sends the new character via UART.

Robustness: CD bits must be correct when communicating. The variable cUrtVar (bits 4 to 6) contains the required CD value.

Side effects: Overwrites ACC, CY, P, RI and TI.
Uses two stack levels.

_getkey

Function prototype: char _getkey(void);

Description of Function: The _getkey function gets a key from UART and flags if error.

User interface: UART must be initialized with UrtCfg() before calling _getkey.
Call _getkey.
_getkey waits for a character to arrive in the UART then returns the character. _getkey sets bit 7 of cUrtVar if a parity error is detected.

Return value: The _getkey function returns a character received by the UART.

Robustness: CD bits must be correct when communicating. The variable cUrtVar (bits 4 to 6) contains the required CD value. Will hang if no character arrives.

Side effects: Overwrites ACC, CY, P, RI and TI.