AN962

Implementing Auto Baud on dsPIC30F Devices

Author: Mike Garbutt
Microchip Technology

INTRODUCTION

All current dsPIC30F devices have a UART peripheral with Auto Baud capability. The signal on the UART receive pin (RX pin) can be internally routed to an Input Capture module to time the edges of the incoming signal. From that timing the software can set up the UART at the correct baud rate.

Auto Baud is useful when the baud rate of the incoming data is unknown and when the oscillator frequency of the processor is unknown. RC oscillators are often inaccurate and drift over time so systems with RC oscillators are ideal candidates for using Auto Baud.

METHOD

The method for doing Auto Baud relies on known data being received. It is usually possible to use a communications protocol that sends data specifically for Auto Baud. The timing of the received data can be used to calculate the value for the U1BRG or U2BRG registers that set the UART baud rates.

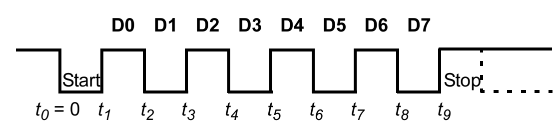
The two examples in this application note use incoming data of 0x55 (ASCII ‘U’) to calculate the baud rate generator value. This particular data byte provides the maximum number of edges and therefore the greatest accuracy. Any data byte can be used and the calculations adapted to suit the data. In general, the more edges (bit state changes) in the data, the more accurate the result.

Signals

UART signals are sent least significant bit first, preceded by a start bit (zero) and followed by a stop bit (one). There are usually eight bits of data, but other sizes are possible, and a parity bit can follow the data. All of this will impact the Auto Baud calculation, but the data should always be known in advance.

There will always be at least two edges because of the start and stop bits, but there could be ten or more. In our example of 0x55 there are ten edges, as shown in Figure 1.

FIGURE 1: UART SIGNAL FOR 0x55



Timing And Sampling

Equation 1 can be used to calculate the value for UxBRG after recording the times of the edges and removing the offset of t_0. This equation is derived in Appendix A using linear regression. The calculation is performed after the last edge has been recorded and should be completed before the start bit of the next byte to ensure that no data is lost. This sets a time limit for the calculation that should be checked. In some cases it will be necessary to use a less complex calculation that can be performed more quickly. Similarly, the time taken for error checking must also be considered.

EQUATION 1: UxBRG LINEAR REGRESSION CALCULATION

$$UxBRG = \frac{2(t_1 + 2t_2 + 3t_3 + \ldots + 9t_9) - 9(t_1 + t_2 + t_3 + \ldots + t_9)}{2640} - 1$$

Errors

It is always good programming practice to check for errors. It is possible that the incoming signal does not contain the expected data of 0x55. The error might be apparent because of a long time period between the edges and a simple time-out can detect this problem. If all the edges occur within the acceptable time-out period, it is still possible that they do not match the expected sequence. There are several statistical methods that can show the degree to which the measured signal deviates from the expected signal.

The mean absolute error has been used in the example code in Appendix B. If the average of all the deviations from the expected time measurements is more than 5% of the bit time, then the data is discarded.

Alternatives

The linear regression method provides excellent accuracy because it uses all the input capture measurements. It may be too slow and computationally intensive for some applications. A simplified method, also derived in Appendix A, is to take the time difference between two edges and divide by the number of bit times to calculate a single bit time.

EQUATION 2: UxBRG SIMPLIFIED CALCULATION

$$UxBRG = \frac{(t_8 - t_0)}{128} - 1$$

The code in appendix C uses this alternate method. Equation 2 uses eight bit times to simplify the calculation because a simple and efficient right shift can be used to divide by a power of two. The example in Appendix C also eliminates some of the error checking for greater speed at the expense of reliability.

CODE EXAMPLES

The code in Appendix B and Appendix C implements Auto Baud on an incoming byte of 0x55 (ASCII ‘U’). The code was developed with the MPLAB® C30 compiler.

Structure

The software has three main parts: the main loop, the initialization and the interrupt routines.

MAIN LOOP

The `main()` function has an endless loop to demonstrate the functioning of the Auto Baud code. It starts by calling `SetupAutoBaud()` to initialize all the peripherals and interrupts used by the Auto Baud procedure. The code then waits until U1BRG has a non-zero value, indicating that the Auto Baud procedure has completed successfully.

The code calculates the actual baud rate from the U1BRG value and sends a message out the UART to show the baud rate. It is not necessary to calculate the baud rate for Auto Baud to work, but it is done for demonstration purposes.

After waiting for the text to be transmitted, the code loops back and starts doing the Auto Baud procedure over again.

INITIALIZATION

The `SetupAutoBaud()` function initializes the UART1 peripheral, Input Capture 1 module and Timer 3 to do Auto Baud on the incoming data.

UART1 is turned on and the Auto Baud feature is enabled. This internally routes the incoming serial data signal on the U1RX pin to be an input to the Input Capture 1 module.

Timer 3 is set up to increment on every instruction cycle for maximum resolution. The period is set to the maximum value so that the timer rolls over after a full 16-bit count.

The Input Capture 1 module is set up to capture Timer 3 and interrupt on every edge of the incoming signal. The Input Capture 1 interrupt is turned on.

INTERRUPT ROUTINES

There are two interrupt routines, one for Timer 3 and one for Input Capture 1.

The Timer 3 interrupt routine counts timer rollover interrupts since the last input capture event. If there is more than one rollover between the edges then the Auto Baud has failed and the procedure starts over. This happens if there is a large gap between the edges of the incoming signal.

The Input Capture 1 interrupt routine is at the heart of the Auto baud procedure. It differs depending on whether the simple calculation or the more complex regression calculation is being done.

The interrupt routine starts by saving the time recorded for the previous edge and reading the new time for the current edge that has been detected. The timer rollover count is reset to zero to start a new time out period.

The first capture interrupt enables the Timer 3 interrupt to check for a time out and initializes the Auto Baud calculation variables.

Each subsequent capture interrupt is used to subtract the current capture time from the previous time to get the time for the current bit. This is done with unsigned integers so that it does not matter whether the timer rolled over between captures.

For the simple calculation the bit time is added to a subtotal until eight bit times have been added together.

For the regression calculation, the bit time is added to the previously recorded time and stored as the next element in an array of times. This provides the points for the regression calculation and the error check later. The two sums needed for regression are also done in the interrupt routine.

When the last (tenth) capture interrupt occurs then the interrupt service routine disables both interrupts, finishes the Auto Baud calculation and enables the UART with the new baud rate.

For the simple calculation there is no error checking, except for the time-outs, and the U1BRG value is determined directly from the sum of eight bit times. The number is rounded off by adding 64 before dividing by 128. This effectively adds ½ before truncating. The division by 128 is done by a seven bit shift

For the regression calculation, the slope and Y intercept of the regression line is calculated. This is then used to calculate the expected times that are subtracted from the actual time measurements for error checking. If the error is more than 5% then the Auto Baud procedure is started over. This threshold can be changed if more or less accuracy is acceptable. Finally, the U1BRG value is calculated from the slope of the regression line.

Using The Code

The example code was developed and tested on a dsPICDEM™ 1.1 board using a dsPIC30F6014 part running at 29.5 MIPS. The timing allows it to be used with any standard baud rate down to 600 baud. The following steps can be used to try out the example code:

- Connect a standard RS232 cable between a COM port on a PC and the connector marked “PORT B” on the dsPICDEM 1.1 board.
- Run a terminal program such as HyperTerminal on the PC. Ensure that the terminal program is using the correct COM port.
- Compile, program and run the code on the dsPIC® device.
- Type “U” in the terminal program.
- The dsPICDEM 1.1 board will respond with the text “Baud rate: xxxx” where xxxx is the baud rate being used.
- Change the baud rate and type “U” in the terminal program again.
- The dsPICDEM 1.1 board will respond with the new baud rate.

Resources Used

The Auto Baud code uses program and data memory, one input capture module and one timer in addition to the UART. Very little RAM is used and the timer and input capture modules can be reused for other purposes after Auto Baud is complete. Program memory usage depends on the sophistication of the calculation. The memory usage shown in Table 1 is the additional memory used by adding the Auto Baud code into the application. All MPLAB C30 applications have a minimum amount of code to handle start-up, initialization, etc. and this has not been included in the figures.

MODIFICATIONS AND IMPROVEMENTS

The code examples provided show two ways to do Auto Baud on an incoming data byte of 0x55 (ASCII ‘U’). These methods can be adapted to any known incoming data. All the methods determine a single bit period by timing the incoming edges. Various degrees of analysis and error checking can be used to improve the reliability or the speed.

It is possible to use Auto Baud on unknown data but it can be difficult to determine a single bit time and to distinguish the start and stop bits from the data bits. Ideally the Auto Baud procedure should use some knowledge of the data in order to simplify the calculations.

The Auto Baud procedure is done in the background by interrupts. The flexible interrupt priority structure of the dsPIC allows the Auto Baud interrupts to be configured to have minimal impact on the rest of the application. The timer and input capture resources used for Auto Baud can be used by the rest of the application after completion of the Auto Baud calculation.

The alternate interrupt vector table can be used for the Auto Baud interrupts, allowing the main application to have its own separate input capture and timer interrupts.

Both UARTs have Auto Baud capability. Note that UART1 uses Input Capture 1 and UART2 uses Input Capture 2 so both UARTs can do Auto Baud at the same time.

CONCLUSION

The built-in Auto Baud feature makes it simple to configure the UART for an unknown baud rate. The process can be done under interrupt control in the background so it has very little impact on the rest of the application. The code can be adapted to user requirements by using a simple calculation for high speed or a complex calculation with sophisticated error checking for better reliability.

REFERENCES

dsPIC30F Family Reference Manual (DS70046)

dsPICDEM™ 1.1 Development Board User’s Guide (DS70099)

MPLAB® C30 C Compiler User’s Guide (DS51284)

TABLE 1: RESOURCES USED BY THE AUTO BAUD CODE

Resource	Simple Calculation	Regression Calculation
Program memory	321 bytes	834 bytes
Data memory	14 bytes	58 bytes
I/O pins	No additional I/O	No additional I/O
Peripherals	Input Capture 1, Timer 3	Input Capture 1, Timer 3
Maximum interrupt execution time	49 TCY	1486 TCY*

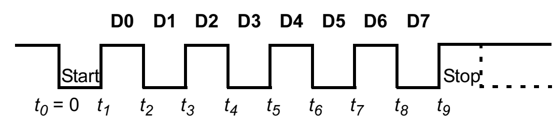
* 618 TCY without mean absolute error calculation

APPENDIX A: CALCULATIONS

REGRESSION CALCULATION

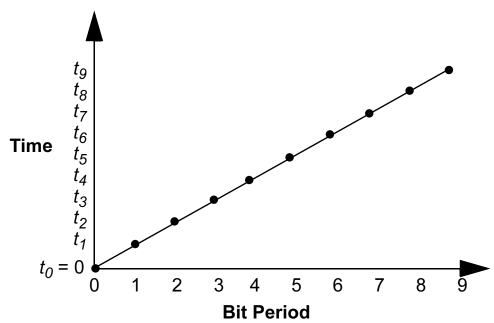
The Input Capture module is used to time the edges of the incoming signal shown in Figure A-1. For a data byte of 0x55, ten times t_0, t_1, t_2… t_9 are collected starting with $t_0 = 0$ as shown in Figure A-1.

FIGURE A-1: UART SIGNAL FOR 0x55



If the incoming signal is correct, the times will each be spaced one bit period apart resulting in the straight line graph in Figure A-2.

FIGURE A-2: GRAPH OF TIME MEASUREMENTS



The slope of the graph is the measured time period per bit. The slope can be calculated by subtracting any two adjacent times. For more accuracy, the first time t_0 can be subtracted from the last time t_9 and divided by the number of bit periods, nine. This will probably be sufficiently accurate for most application but still only uses two of the time measurements.

Linear regression can be used to get the most accurate representation of the slope of the graph. This is a statistical technique that finds the best fit of a line though a set of points. Equation A-1 is the linear regression calculation for the slope *m* of a line through *n* points (x_0,y_0), (x_1,y_1), (x_2,y_2)… (x_n,y_n).

EQUATION A-1: LINEAR REGRESSION EQUATION

$$m = \frac{n\sum(xy) - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$$

Substitute 0, 1, 2… 9 for x and t_0, t_1, t_2… t_9 for y in this equation. All the values of x are known, $t_0 = 0$ and n = 10 so the equation reduces to Equation A-2.

The simplified equation requires ten multiplications and one division which are easily done on the dsPIC30F devices.

EQUATION A-2: TIME PER BIT PERIOD

$$m = \frac{10(t_1 + 2t_2 + 3t_3 + \ldots + 9t_9) - 45(t_1 + t_2 + t_3 + \ldots + t_9)}{825}$$

$$= \frac{2(t_1 + 2t_2 + 3t_3 + \ldots + 9t_9) - 9(t_1 + t_2 + t_3 + \ldots + t_9)}{165}$$

UxBRG CALCULATION

The slope *m* of the line is the number of timer counts per bit period *p*. If we set up the timer to count instruction cycles TCY (TCY = 1/FCY) then the bit period *p* is expressed in Equation A-3. The reciprocal of the bit period is the baud rate as shown in Equation A-4.

EQUATION A-3: BIT PERIOD

$$p = m \times Tcy$$

EQUATION A-4: MEASURED BAUD RATE

$$baud = \frac{1}{p} = \frac{1}{m \times Tcy} = \frac{Fcy}{m}$$

The baud rate of the UART is controlled by setting the UxBRG register. Equation A-5 shows the baud rate in terms of UxBRG, as expressed in the *dsPIC30F Family Reference Manual*.

EQUATION A-5: SPECIFIED BAUD RATE

$$baud = \frac{Fcy}{16(UxBRG + 1)}$$

Combining Equation A-4 and Equation A-5 gives the UxBRG value in terms of the slope *m* that is calculated from the input capture times t_0, t_1, t_2… t_9.

Equation A-6 results:

EQUATION A-6: UxBRG VALUE

$$UxBRG = \frac{m}{16} - 1$$

Equation A-2 and Equation A-6 combine to provide a single calculation for the UxBRG register value that corresponds to the incoming baud rate. This is shown in Equation A-7.

EQUATION A-7: COMPLETE CALCULATION

$$UxBRG = \frac{2(t_1 + 2t_2 + 3t_3 + \ldots + 9t_9) - 9(t_1 + t_2 + t_3 + \ldots + t_9)}{2640} - 1$$

Simple UxBRG Calculation

Where speed is critical the bit time, which is the slope *m* of the signal, can be calculated very quickly from two time measurements as shown in Equation A-8 below.

EQUATION A-8: BIT TIME SIMPLIFIED CALCULATION

$$m = \frac{(t_8 - t_0)}{8}$$

This equation uses the times of two falling edges in order to cancel out differences in rise and fall times and propagation delays. Averaging eight bit periods also makes the division easier because a simple right shift can be used to divide. Combining Equation A-6 and Equation A-8 results in the final simplified calculation for UxBRG given in Equation A-9 below.

EQUATION A-9: UxBRG SIMPLIFIED CALCULATION

$$UxBRG = \frac{(t_8 - t_0)}{128} - 1$$

Error Calculation

There are several statistical techniques to express the deviation of measured results from their expected values. The correlation r or the standard deviation δ can be calculated but this is computationally intensive and time consuming because there are several square functions and a square root. A simpler method is the mean absolute error, an average of the absolute values of the errors as shown in Equation A-10.

EQUATION A-10: MEAN ABSOLUTE ERROR

$$MAE = \frac{1}{n}\sum |x - \bar{x}|$$

The symbol $\bar{x}$ is the expected value of the measured signal. We can determine the expected values of all the times t_0, t_1, $t_2 \ldots t_9$ from the graph in Figure A-2. The slope *m* of the graph has been calculated in Equation A-2 and the y intercept *b* can be calculated from Equation A-11.

EQUATION A-11: Y INTERCEPT

$$b = \frac{\sum y - m\sum x}{n}$$

Substitute 0, 1, 2… 9 for *x* and t_0, t_1, $t_2 \ldots t_9$ for *y* in this equation. All the values of *x* are known, $t_0 = 0$ and n =10 so the equation reduces to Equation A-12. In this case the y intercept is actually the expected value of t_0.

EQUATION A-12: EXPECTED VALUE OF Y INTERCEPT

$$\dot{t}_0 = \frac{(t_1 + t_2 + t_3 + \ldots + t_9) - 45m}{10}$$

This calculation is very easy to perform because it only uses one multiplication and division and $(t_1 + t_2 + \ldots + t_9)$ has already been calculated in Equation A-2 to get the slope.

Once the slope and intercept are known, all the expected values can be calculated from the equation of the line, Equation A-13.

EQUATION A-13: EXPECTED TIMES

$$\dot{t}_i = mx_i + \dot{t}_0$$

Substituting 0, 1, 2… 9 for x_0, x_1, $x_2 \ldots x_9$ provides all the expected values. In software it is easier to add the slope *m* to each result to get the next value because the values of x_i increment by one.

APPENDIX B: SOURCE CODE USING REGRESSION CALCULATION


```
/*********************************************************************
 *
 *              dsPIC30F Auto Baud Source Code
 *
 *********************************************************************
 * FileName:       UART Auto Baud by Regression.c
 * Dependencies:   p30F6014.h
 *                 math.h
 * Date:           10/08/2004
 * Processor:      dsPIC30F6014
 * Complier:       MPLAB C30 1.20.02
 * Company:        Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its PICmicro® Microcontroller is intended and
 * supplied to you, the Company's customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Author              Date       Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Mike Garbutt       10/8/2004   Original       (Rev 1.0)
 *
 ********************************************************************/

#include "p30F6014.h"            //Standard header file
#include "math.h"                //Math library
#define Fcy  29491200            //To allow calculation of baud rate for display

//---------------------------------------------------------------------------
//Prototypes

void SetupAutoBaud(void);        //Function to set up UART1, IC1 and TMR3
void CalculateBaud(void);        //Function to calculate the U1BRG value
```



```
//---------------------------------------------------------------------------
//Variables

unsigned int ICCount = 0;                    //Count the number of Capture events
unsigned int T3Count = 0;                    //Count the number of Timer 3 interrupts
unsigned int CurrentCapture;                 //Record time of UART edge
unsigned int PreviousCapture;                //Store previous edge time measurement
unsigned int CaptureDifference;              //Difference between times of UART edges
long SumXY, SumY;                            //Intermediate value for regression calculation
long RegressionData[10];                     //Data to do linear regression
unsigned long BaudRate;                      //Calculate the baud rate

//---------------------------------------------------------------------------
//Main routine
//Loops forever detecting the baud rate from incoming UART data of 0x55
//and outputting a message each time the baud rate is calculated.

int main(void)
{
    while(1)                                     //Loop forever
    {
        SetupAutoBaud();                         //Set up UART1, IC1 and TMR3 for autobaud
        while(U1BRG == 0) {}                     //Wait for autobaud to complete
        BaudRate = (Fcy / 16) / (U1BRG + 1);     //See what baud rate is being used
        printf("Baud rate: %ld\r", BaudRate);    //Output text with the baud rate
    while(U1STAbits.TRMT == 0) {}                //Wait for transmission to complete
    }
}                                                //End of main()

//---------------------------------------------------------------------------
//Set up the peripherals and interrupts to do baud rate detection

void SetupAutoBaud(void)
{
    U1BRG = 0;                                   //U1BRG initially unknown
    U1MODE = 0x8020;                             //Enable auto baud detection in UART
    U1STA = 0x0000;                              //Set up rest of UART to default state

    ICCount = 0;                                 //Initialize the number of Capture events
    IC1CON = 0x0000;                             //Reset Input Capture 1 module
    IC1CON = 0x0001;                             //Enable Input Capture 1 module
    IFS0bits.IC1IF = 0;                          //Clear Capture 1 interrupt flag
    IEC0bits.IC1IE = 1;                          //Enable Capture 1 interrupt

    T3CON = 0x0000;                              //Timer 3 off
    IEC0bits.T3IE = 0;                           //Clear Timer 3 interrupt enable
    T3Count = 0;                                 //Initialize the number of Timer 3 interrupts
    PR3 = 0xffff;                                //Timer 3 period is maximum
    T3CON = 0x8000;                              //Timer 3 on with 1:1 prescaler and internal clock
}

//---------------------------------------------------------------------------
//Calculate value for U1BRG baud rate generator

void CalculateBaud(void)
{
    int i;                                       //Index to sum the errors
    long Slope;                                  //Slope (bit time) of regression
    long Yintercept;                             //Expected (calculated) time of first edge
    long PlotLine;                               //Expected (calculated) time of each edge
    long SumOfErrors = 0;                        //Sum of all the errors |Measured-Expected|

    Slope = (2 * SumXY - 9 * SumY) / 165;        //Calculate slope = one bit time
    Yintercept = (SumY - Slope * 45) / 10;       //Calculate expected time of first edge
```

```
    PlotLine = Yintercept;                          //Initialize expected time of each edge
    for(i=0; i<10; i++)                             //Loop to add all the absolute errors
    {
        SumOfErrors += abs(RegressionData[i] - PlotLine);//Calculate and add next error
        PlotLine += Slope;                                  //Calculate expected time of next edge
    }

    if((SumOfErrors * 2) < Slope)                   //Check if mean absolute error < 5%
    {                                               //(is twice sum of 10 errors < one bit time?)
        U1BRG = ((Slope + 8) >> 4) - 1;             //Calculate UxBRG (rounding by adding one half)
        U1MODE = 0x8000;                            //Enable UART and disable auto baud detection
        U1STA = 0x0400;                             //Enable transmission
    }
    else
    {
    SetupAutoBaud();                                //Error too large so start over
    }
}
//---------------------------------------------------------------------------
//Input Capture 1 ISR
//Gets time measurements and adds to the sums needed for regression

void _ISR _IC1Interrupt(void)
{
    IFS0bits.IC1IF = 0;                         //Clear Capture 1 interrupt flag
    PreviousCapture = CurrentCapture;           //Store previous time measurement
    CurrentCapture = IC1BUF;                    //Get new time measurement
    T3Count = 0;                                //Reset the timeout counter
    if(ICCount == 0)                            //Check if first edge
    {
        IFS0bits.T3IF = 0;                      //Clear Timer 3 interrupt flag
        IEC0bits.T3IE = 1;                      //Enable Timer 3 interrupt for timeout check
        RegressionData[0] = 0;                  //Initial value for time of first edge
        SumY = 0;                               //Initial value for sum of time measurements
        SumXY = 0;                              //Initial value for sum of bit number x time
    }
    else                                        //Check if not first edge
    {
        CaptureDifference = CurrentCapture - PreviousCapture; //Get time difference
        RegressionData[ICCount] = RegressionData[ICCount-1] + CaptureDifference;
                                                //Add time difference to prevous time measurement
        SumY += RegressionData[ICCount];        //Sum the time measurements
        SumXY += RegressionData[ICCount] * ICCount;//Sum the bit number x time measurement
    }
    ICCount++;                                  //Increment count of edges
    if(ICCount == 10)                           //Check if last edge
    {
        IEC0bits.IC1IE = 0;                     //Clear Capture 1 interrupt enable
        IEC0bits.T3IE = 0;                      //Clear Timer 3 interrupt enable
        CalculateBaud();                        //Calculate the U1BRG value and enable the UART
    }
}
//---------------------------------------------------------------------------
//Timer 3 ISR
//Check for timeout indicated by two rollovers since previous input capture

void _ISR _T3Interrupt(void)
{
   IFS0bits.T3IF = 0;              //Clear Timer 3 interrupt flag
   T3Count++;                      //Increment count of interrupts since last capture
   if(T3Count == 2)                //Check for too many timer rollovers since last capture
   {
      SetupAutoBaud();             //Timeout so start over
   }
}
```


APPENDIX C: SOURCE CODE USING SIMPLE CALCULATION

```
/*********************************************************************
 *
 *              dsPIC30F Auto Baud Source Code
 *
 *********************************************************************
 * FileName:        UART Auto Baud by Simple Calculation.c
 * Dependencies:    p30F6014.h
 *                  math.h
 * Date:            10/08/2004
 * Processor:       dsPIC30F6014
 * Complier:        MPLAB C30 1.20.02
 * Company:         Microchip Technology, Inc.
 *
 * Software License Agreement
 *
 * The software supplied herewith by Microchip Technology Incorporated
 * (the "Company") for its PICmicro® Microcontroller is intended and
 * supplied to you, the Company's customer, for use solely and
 * exclusively on Microchip PICmicro Microcontroller products. The
 * software is owned by the Company and/or its supplier, and is
 * protected under applicable copyright laws. All rights are reserved.
 * Any use in violation of the foregoing restrictions may subject the
 * user to criminal sanctions under applicable laws, as well as to
 * civil liability for the breach of the terms and conditions of this
 * license.
 *
 * THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
 * WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
 * TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
 * PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
 * IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
 * CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 * Author             Date           Comment
 *~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 * Mike Garbutt       10/8/2004      Original        (Rev 1.0)
 *
 ********************************************************************/

#include "p30F6014.h"                //Standard header file
#include "math.h"                    //Math library
#define Fcy  29491200                //To allow calculation of baud rate for display

//-----------------------------------------------------------------------------
//Prototypes

void SetupAutoBaud(void);            //Function to set up UART1, IC1 and TMR3
void CalculateBaud(void);            //Function to calculate the U1BRG value

//-----------------------------------------------------------------------------
//Variables

unsigned int ICCount = 0;            //Count the number of Capture events
unsigned int T3Count = 0;            //Count the number of Timer 3 interrupts
unsigned int CurrentCapture;         //Record time of UART edge
unsigned int PreviousCapture;        //Store previous edge time measurement
unsigned int CaptureDifference;      //Difference between times of UART edges
unsigned long SumOfBitTimes;         //
unsigned long BaudRate;              //Calculate the baud rate
```

```
//---------------------------------------------------------------------------
//Main routine
//Loops forever detecting the baud rate from incoming UART data of 0x55
//and outputing a message each time the baud rate is calculated.

int main(void)
{
    while(1)                                  //Loop forever
    {
        SetupAutoBaud();                      //Set up UART1, IC1 and TMR3 for autobaud
        while(U1BRG == 0) {}                  //Wait for autobaud to complete
        BaudRate = (Fcy / 16) / (U1BRG + 1);  //See what baud rate is being used
        printf("Baud rate: %ld\r", BaudRate); //Output text with the baud rate
        while(U1STAbits.TRMT == 0) {}         //Wait for transmission to complete
    }
}                                             //End of main()
//---------------------------------------------------------------------------
//Set up the peripherals and interrupts to do baud rate detection

void SetupAutoBaud(void)
{
    U1BRG = 0;                                //U1BRG initially unknown
    U1MODE = 0x8020;                          //Enable auto baud detection in UART
    U1STA = 0x0000;                           //Set up rest of UART to default state

    ICCount = 0;                              //Initialize the number of Capture events
    IC1CON = 0x0000;                          //Reset Input Capture 1 module
    IC1CON = 0x0001;                          //Enable Input Capture 1 module
    IFS0bits.IC1IF = 0;                       //Clear Capture 1 interrupt flag
    IEC0bits.IC1IE = 1;                       //Enable Capture 1 interrupt

    T3CON = 0x0000;                           //Timer 3 off
    IEC0bits.T3IE = 0;                        //Clear Timer 3 interrupt enable
    T3Count = 0;                              //Initialize the number of Timer 3 interrupts
    PR3 = 0xffff;                             //Timer 3 period is maximum
    T3CON = 0x8000;                           //Timer 3 on with 1:1 prescaler and internal clock
}
//---------------------------------------------------------------------------
//Input Capture 1 ISR
//Gets time measurements and adds to the sum of the bit times
//Calculates U1BRG value after summing eight bit times

void _ISR _IC1Interrupt(void)
{
    IFS0bits.IC1IF = 0;                       //Clear Capture 1 interrupt flag
    PreviousCapture = CurrentCapture;         //Store previous time measurement
    CurrentCapture = IC1BUF;                  //Get new time measurement
    T3Count = 0;                              //Reset the timeout counter
    if(ICCount == 0)                          //Check if first edge
    {
        IFS0bits.T3IF = 0;                    //Clear Timer 3 interrupt flag
        IEC0bits.T3IE = 1;                    //Enable Timer 3 interrupt for timeout check
        SumOfBitTimes = 0;                    //Initial value for sum of the bit times
    }
    else                                      //Check if not first edge
    {
        if(ICCount != 9)                      //Check if not last edge
        {
            CaptureDifference = CurrentCapture - PreviousCapture; //Get time difference
            SumOfBitTimes += CaptureDifference; //Add time difference to sum of times
        }
    else                                      //Check if last edge
        {
            IEC0bits.IC1IE = 0;               //Clear Capture 1 interrupt enable
            IEC0bits.T3IE = 0;                //Clear Timer 3 interrupt enable
```

```
            U1BRG = ((SumOfBitTimes + 64) >> 7) - 1; //Calculate UxBRG (with rounding)
            U1MODE = 0x8000;              //Enable UART and disable auto baud detection
            U1STA = 0x0400;               //Enable transmission
        }
    }
    ICCount++;                            //Increment count of edges
}
//-----------------------------------------------------------------------------
//Timer 3 ISR
//Check for timeout indicated by two rollovers since previous input capture

void _ISR _T3Interrupt(void)
{
    IFS0bits.T3IF = 0;                    //Clear Timer 3 interrupt flag
    T3Count++;                            //Increment count of interrupts since last capture
    if(T3Count == 2)                      //Check for too many timer rollovers since last capture
    {
        SetupAutoBaud();                  //Timeout so start over
    }
}
```

NOTES:

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

Boston
Westford, MA
Tel: 978-692-3848
Fax: 978-692-3821

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

San Jose
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Qingdao
Tel: 86-532-502-7355
Fax: 86-532-502-7205

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

Japan - Kanagawa
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Taiwan - Hsinchu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

EUROPE

Austria - Weis
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393

Denmark - Ballerup
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Massy
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Ismaning
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

England - Berkshire
Tel: 44-118-921-5869
Fax: 44-118-921-5820

10/20/04