Automatic Baud Rate Detection and Configuration in UCD3138

Emily Hirsch *High Performance Isolated Power*

Abstract

When there are major changes in temperature, the UCD3138 internal oscillator frequency can change. See the datasheet for more specific information. There can be a need to implement an automatic baud rate detection and configuration to improve serial communication between the primary and secondary stages of a power supply. This application report discusses how to configure the UCD3138 to match its baud rate to an incoming serial data stream.

Contents

1 Introduction

When using UCD3138s for PFC and LLC applications with serial port communication between the two, there can be a slight difference in the two oscillators due to temperature. To compensate for this variation and prevent errors in primary to secondary UART communication, an automatic baud rate matching solution can be implemented into either PFC or LLC. The PFC (primary side) and LLC (secondary side) constantly transmit information to each other such as fault status, operational modes, voltages, and currents. The information used in this solution is transmitted through UART0 in the PFC and received through UART0 in the LLC. The solution implements a Timer Capture task inside of the 10kHz timer interrupt to measure the receive baud rate because we can catch both edges of a 4800 single byte pulse. The solution captures one, two, three, or four bit long pulses and configures the appropriate UART registers in the LLC to match the baud rate of the PFC.

2 Timer Capture

2.1 Overview

Timer Capture is implemented using a 24 bit free-running timer in the UCD3138. The 24 bit timer is configured to capture edges on the UART receive pin. The measure_baud() function will monitor the waveform. Whenever an edge is captured, the value of the 24 bit counter is captured and put into the capture register, which is read on every edge of the waveform. On the first edge (falling edge) the UCD3138 stores the value of the 24 bit counter and on the second edge (rising edge) it takes the value from the 24 bit counter and subtracts the value found from the first edge to get the pulse width of the signal. The negative pulse width is capture because the idle state of the UART is high, so a short time of idle could seem like a pulse width and the wrong baud rate could be calculated. Figure 2 below represents capturing the positive and negative edges of a waveform.

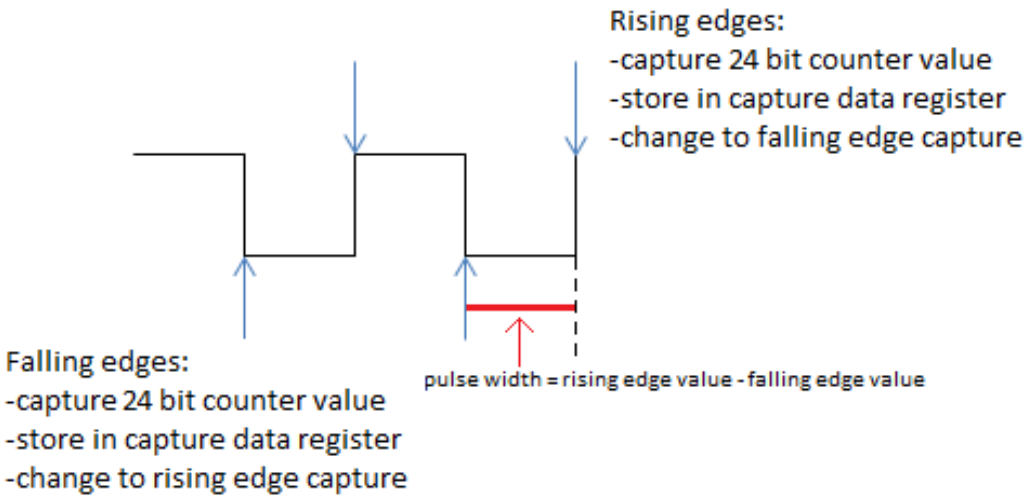

Figure 2: Timer Capture on a Waveform

The clock input for the timer is not prescaled with this application and is solely the ICLK, running at about 15.6MHz. This method uses polling inside of the interrupt service routine to capture edges, so every 10kHz (the frequency of the 16 bit timer interrupt) the measure baud rate sequence will be executed using timer capture. This solution works for 4800 bps and below because the bit time is less than 5KHz with the interrupt at 10 KHz, so we can capture each edge with a different interrupt. Faster baud rates than 4800 would require a faster interrupt or the timer capture interrupt. Only the capture block, not the compare block, for the 24 bit timer is used in this application to collect information about the pulse width of the signal received on the LLC. There is more information on Timer Capture in Section 4 of the Monitoring and Communications Programmer's Manual (Literature Number: SLU996).

2.2 Timer Capture Registers

A few registers for Timer Capture must be set for this application. The input signal for Timer Capture must be set to the SCI_RX0 pin. In order to ensure that the negative pulse width is being calculated, the edge must originally be set to capture on the falling edge of the input signal. This will be changed later in the code so it only captures the negative pulse width. The registers that must be set, along with the values they need to be set to are shown below:

```
TimerRegs.T24CAPCTRL.bit.CAP_SEL = 1; //input signal comes from SCI_RX0 pin
TimerRegs.T24CAPCTRL.bit.EDGE = 2;    //enable capture on falling edge
```

2.3 Timer Capture Polling

The timer capture polling can be integrated into the standard interrupt in the LLC code. Every time the timer interrupt is triggered, all global tasks are handled, and the measure baud function has been included in the global tasks along with collecting ADC data, averaging ADC readings, receiving UART data, and calling the pgood handler. The full code is shown in Section 6 Installing Code into Existing Program. Whenever the first falling edge has been captured, the capture interrupt flag is set and the value of the 24 bit counter is stored into a data register for the capture module. In order to get the negative pulse length, the edge must be switched to capture the rising edge. The two edges are captured and the value captured at the falling edge must be subtracted from the value captured at the rising edge. The measure_baud() function is included below:

```
void measure_baud(void)
{
      T24SREG = TimerRegs.T24CAPCTRL.bit.CAP_INT_FLAG;
      if((T24SREG == 1) && (edge == 0))         //first edge detected
      {
            result = TimerRegs.T24CAPDAT.bit.CAP_DAT; //read and clear
                  //enable capture on rising edge
            TimerRegs.T24CAPCTRL.bit.EDGE = 1;
            edge = 1;                           //capture second edge
      }
      else if((T24SREG == 1) && (edge == 1))    //second edge detected
      {
                  //read and clear register value
            result = TimerRegs.T24CAPDAT.bit.CAP_DAT – result;
                  //result contains pulse width
            pulse_width = result;               //store result in pulse_width
                  //enable capture on falling edge
```

```
            TimerRegs.T24CAPCTRL.bit.EDGE = 2;
            edge = 0;               //reset edge to capture first edge again
      }
      else
      {
            //do nothing
      }
}
```

By reading the CAP_INT_FLAG register into the variable “T24SREG,” the UCD3138 clears the interrupt flag and stores the flag data into T24SREG. The variable “edge” determines which edge (first or second) is being captured. “Result” stores the value captured and puts it into the data register. This result is then stored into “pulse_width” and used in a different function afterwards to change the baud rate.

2.3.1 Number of Bits Captured

The timer capture interrupt will continually capture pulse widths for every bit sent, so it is important to understand what bits of each byte could be captured and how long of a pulse the timer capture will measure. The message sent is from LSB to MSB, which will be explained further in Section 4.3 UART Byte Format, and the number of bits captured is important because the baud rate will only be calculated if one, two, three, or four bits are captured. Table 1 below shows what one, two, three, four, and five bit pulses looks like from the UART or Timer Capture. As shown, a one bit pulse width is captured whenever there is a one, followed by one zero, then a one afterward. A two bit pulse is the same except there are two zeroes, a three bit pulse has three zeroes, and a four bit pulse has four zeroes.

Table 1: Number of Bits Captured for Certain Messages

Value sent (as seen in code)	Message Sent (as seen by UART)	Entire Message Sent with start and stop bits (as seen by UART)	Possible Number of Bits Captured
Hexadecimal:0x49 Binary:01001001	Hexadecimal: 0x92 Binary: 10010010	01001001011	1 or 2 match baud possible
Hexadecimal: 0xFD Binary: 11111101	Hexadecimal: 0xBF Binary: 10111111	01011111111	1 match baud possible
Hexadecimal: 0x35 Binary: 00110101	Hexadecimal: 0xAC Binary: 10101100	01010110011	1 or 2 change baud possible
Hexadecimal: 0xC8 Binary: 11001000	Hexadecimal: 0x13 Binary: 00010011	00001001111	2 or 4 match baud possible
Hexadecimal: 0x1A Binary: 00011010	Hexadecimal: 0x58 Binary: 01011000	00101100011	1, 2, or 3 match baud possible

Hexadecimal: 0xE1 Binary: 11100001	Hexadecimal: 0x87 Binary: 10000111	01000011111	1 or 4 match baud possible
Hexadecimal: 0x83 Binary: 10000011	Hexadecimal: 0xC1 Binary: 11000001	01100000111	1 or 5 match baud possible
Hexadecimal: 0xF0 Binary: 11110000	Hexadecimal: 0x0F Binary: 00001111	00000111111	5 match baud impossible

If there are a variety of bytes being sent with at least a majority of them containing a one to four bit pulse inside, then there will be a robust form of communication between the primary and secondary side. With this solution, however, if it does not send any characters that will give a one to four bit pulse in the byte, this specific automatic baud change application will not work for it.

3 UART

3.1 Overview

There are two UARTs, or universal asynchronous receiver-transmitters, in UCD3138. These UARTs are used to transmit information between the primary and secondary sides of the power supply. For this solution, UART0 is used on both the primary and secondary sides to send and receive bytes, and the bytes transmitted are used to capture pulse widths of the primary side. Baud rates are set using the baud divide registers and on the secondary side, they will be changed after initialization to match the baud rate of the primary side. The received bytes come in to the data register in the UART receive buffer and can be viewed on the UART0 receive pin of UCD3138. The transmitted bytes transmit from the UART transmit buffer and can be viewed on the UART0 transmit pin of UCD3138. More information on UARTs in the UCD3138 can be found in Section 5 of the Monitoring and Communications Programmer's Manual.

3.2 UART0 Baud Divide Registers

The baud rate is adjusted using the baud divide registers. There are three divide registers for each UART, a low divide, middle divide, and high divide (2).The overall register value is found using Equation 1 shown below, where "register value" represents the overall value that will be split into low, middle, and high divide values; "ICLK" represents the internal 15.625MHz clock; "baud" represents the desired baud rate. After determining the overall register value, the low divide value is found by converting the register value from decimal to hexadecimal, then AND it with 0xFF. The middle divide value is found by shifting the hexadecimal register value to the right 8 bits. The high divide value is always zero for this application. At baud rates higher than 38400, the step sizes for the baud divide values are larger and may not be sufficient to use this type of solution for automatic baud detection and matching.

$$register\ value = \frac{ICLK}{8 * baud} - 1$$

Equation 1: Register value equation [3]

For example, the divide values for a baud rate of 4800 bps are configured using either of these two sets of statements:

Example 1 [3]:

```
//BAUD_RATE set to 4800 bps, BAUD_RATE_VALUE and divide values
            //are calculated in code, rather than by hand

#define BAUD_RATE                   4800
#define BAUD_RATE_VALUE       (15625000/ (8* (BAUD_RATE + 1) ) )
#define BAUD_RATE_VALUE_M     (BAUD_RATE_VALUE & 0xFF)
#define BAUD_RATE_VALUE_L     (BAUD_RATE_VALUE >> 8)

Uart0Regs.UARTHBAUD.bit.BAUD_DIV_H = 0;       //always zero for our purposes
Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = BAUD_RATE_VALUE_M;
Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = BAUD_RATE_VALUE_L;
```

Example 2:

```
      //baud divide values calculated by hand and input directly into code
Uart0Regs.UARTHBAUD.bit.BAUD_DIV_H = 0;   //always zero for our purposes
Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = 1;   //overall register value shifted
right 8 bits
Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = 149; //overall register value AND 0xFF
```

3.3 UART Byte Format

For this application, when a message is sent through UART, it contains a start bit, 8 bits of data and two stop bits, shown in Figure 3. The idle state is high, so the start bit is always low and the two stop bits are always high, with the message in between. The message is sent from Least Significant Bit to Most Significant Bit, so UART0 on the LLC receives the bits from LSB to MSB.

Figure 3: UART Message Format [2]

4 Baud Rate Matching Firmware

4.1 Timer Value to Baud Rate Translation

After the timer capture interrupt has captured both edges of a pulse, it has a pulse width for an unknown number of bits between one and nine bits. This bit time is then sent into a function to change the baud rate for both of the UARTs in the LLC. In this function, even though it changes the baud rate, the baud rate is never actually calculated in order to save time and memory. Instead, the formula for translating the bit time to baud rate (Equation A) and the baud rate to the overall baud value (Equation B) is shortened to an equation taking the bit time and finding the

overall baud value (Equation C or D) in the UART0 register, shown below in Figure 4. After finding the register value from Equation D, the middle and low divide values are found using the method explained in Section 4.2 UART0 Baud Divide Registers.

$$\text{Equation A:} \quad pulse = \frac{ICLK}{baud}$$

$$\text{Equation B:} \quad register\ value = \frac{ICLK}{8 * baud} - 1$$

$$\text{Equation C:} \quad register\ value = \frac{pulse}{8} - 1$$

$$\text{Equation D:} \quad register\ value = (pulse \gg 3) - 1$$

Figure 4: Equations used to Find Register Values [3]

This final equation (Equation D) can only be used if one bit pulse was captured. If more than one bit pulse was captured, there has to be a compensation to account for the multiple bits acquired.

4.2 Different Bit Pulse Widths

For this application, we only change the baud rate if one, two, or three bit pulses have been captured and we assume the baud rate will be between +/- 10% from 4800 bps. If the pulse length is longer than expected for 3 bits at the slowest baud rate or shorter than 1 bit at the fastest baud rate, the baud rate isn't changed and the information is treated as an error, or garbage. A block diagram of the changing baud function is shown in Figure 5.

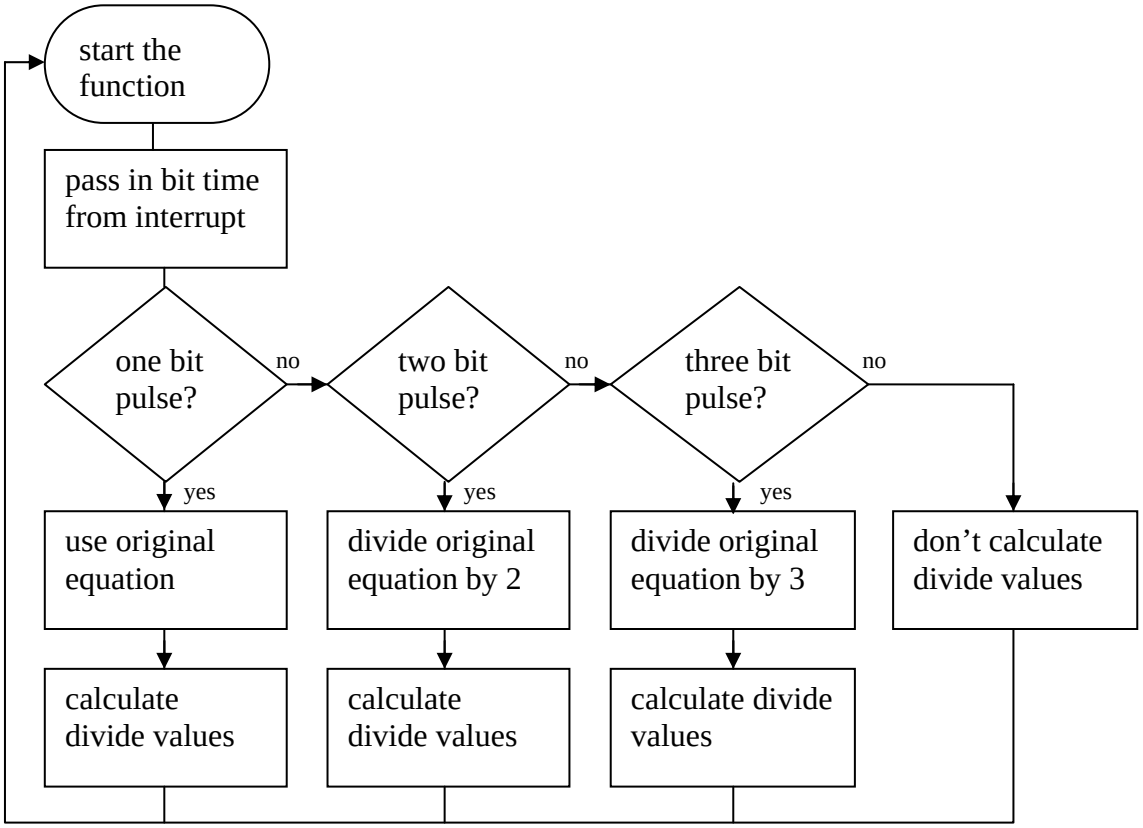

Figure 5: Flowchart of Baud Rate Configuration

4.2.1 One Bit Pulse

As mentioned before, if a one bit pulse is captured, Equation D from Figure 4 in Section 5.1 is the function used to calculate the baud divide values from the bit time. Since a one bit pulse is an exact interpretation of the equations in Figure 4, pulse width times, called "bit_time" in the code, are calculated using Equation A from Figure 4. The code used to calculate the baud divide values when a one bit pulse has been found is included below:

```
        //calculated correct pulse width (+/- 10% from 4800 baud)
if ((bit_time >= 2950) && (bit_time <= 3260))
{
        baud_div_value = (bit_time >> 3) - 1;
        Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
        Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
        Uart1Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
        Uart1Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
}
```

As shown in the code, the bit times for a one bit pulse must be between 2950 and 3260 in order to calculate the baud divide values. This was determined by taking a baud rate of 4800 bps and calculating what baud rates would be +/- 10% from 4800 bps and converting that to the bit time. Table 2.1 below shows the bit time, baud rate, and percent difference from 4800 bps.

Table 2.1: One Bit Pulse Bit Time and Baud Rate

bit time	corresponding baud	baud with adjustment	+/- from 4800
2950	5296	5296	+10.33%
3255	4800	4800	0.00%
3620	4316	4316	-10.08%

4.2.2 Two Bit Pulse

If a two bit pulse is captured, Equation D from Figure 4 in Section 5.1 needs to be modified in order to account for the longer pulse width captured. Since the time captured is twice the actual time for one bit, the equation must be divided by two. The equation and code used to calculate the baud divide values when a two bit pulse has been found are included below:

Equation C Modified:

$$register\ value = \frac{pulse}{16} - 1$$

Equation D Modified:

$$register\ value = ((pulse \gg 4) \& 0xffff) - 1$$

Code:

```
        //calculated 2 times pulse width (+/- 10% from 4800 baud)
```

```
if ((bit_time >= 5900) && (bit_time <= 7240))
{
      baud_div_value = ((bit_time >> 4)&0xffff) - 1;
      Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
      Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
      Uart1Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
      Uart1Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
}
```

As shown in the code, the bit times for a two bit pulse must be between 5900 and 7240 in order to calculate the baud divide values. This was determined by taking a baud rate of 4800 bps and determining what baud rates would be +/- 10% from 4800 bps. Those rates must then be divided by 2 in order to get the "baud rate" that would be calculated for a two bit pulse and converting that to the bit time. Table 2.2 below shows the bit time, calculated baud rate, baud rate adjusted to be centered around 4800, and percent difference from 4800 bps.

Table 2.2: Two Bit Pulse Bit Time and Baud Rate

bit time	corresponding baud	baud with adjustment	+/- from 4800
5900	2648	5296	+10.33%
6510	2400	4800	0.00%
7240	2158	4316	-10.08%

4.2.3 Three Bit Pulse

If a three bit pulse is captured, Equation D is divided by three in order to account for capturing three times the pulse width centered at 4800 bps. The modified equation and code used to calculate the baud divide values when a three bit pulse has been found are included below:

Equation C Modified:

$$register\ value = \frac{pulse}{24} - 1$$

Equation D Modified:

$$register\ value = ((pulse * 2730) \gg 16) - 1$$

Code:

```
      //calculated 3 times pulse width (+/- 10% from 4800 baud)
if ((bit_time >= 8850) && (bit_time <= 10860))
{
      baud_div_value = ((bit_time * 2730) >> 16) - 1;
      Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
      Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
      Uart1Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
      Uart1Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
}
```

As shown in the code, the bit times for a three bit pulse must be between 8850 and 10860 in order to calculate the baud divide values. This was determined by taking a baud rate of 4800 bps and calculating what baud rates would be +/- 10% from 4800 bps. Those rates must then be

divided by 3 in order to get the “baud rate” that would be calculated for a three bit pulse and converting that to the bit time. Table 2.3 below shows the bit time, calculated baud rate, baud rate adjusted to be centered around 4800, and percent difference from 4800 bps.

Table 2.3: Three Bit Pulse Bit Time and Baud Rate

bit time	corresponding baud	baud with adjustment	+/- from 4800
8850	1765	5295	+10.31%
9765	1600	4800	0.00%
10860	1438	4314	-10.13%

4.2.4 Four Bit Pulse

If a four bit pulse is captured, Equation D is divided by four in order to account for capturing four times the pulse width centered at 4800 bps. The modified equation and code used to calculate the baud divide values when a four bit pulse has been found are included below:

Equation C Modified:

$$register\ value = \frac{pulse}{32} - 1$$

Equation D Modified:

$$register\ value = (pulse >> 5) - 1$$

Code:

```
        //calculated 4 times pulse width (+/- 10% from 4800 baud)
if ((bit_time >= 11801) && (bit_time <= 14481))
{
        baud_div_value = (bit_time >> 5) - 1;
        Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
        Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
        Uart1Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
        Uart1Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
}
```

As shown in the code, the bit times for a four bit pulse must be between 11801 and 14481 in order to calculate the baud divide values. This was determined by taking a baud rate of 4800 bps and calculating what baud rates would be +/- 10% from 4800 bps. Those rates must then be divided by 4 in order to get the “baud rate” that would be calculated for a four bit pulse and converting that to the bit time. Table 2.4 below shows the bit time, calculated baud rate, baud rate adjusted to be centered around 4800, and percent difference from 4800 bps.

Table 2.4: Four Bit Pulse Bit Time and Baud Rate

bit time	corresponding baud	baud with adjustment	+/- from 4800
11801	1324	5296	+10.33%
13020	1200	4800	0.00%
14481	1079	4316	-10.08%

4.2.5 Five or More Bit Pulse

If a five bit pulse or longer has been captured, the baud rate will not match to the PFC because going higher than four bits creates an overlap in the bounds between -10% from 4800 with a 5 bit pulse and +10% from 4800 with a 6 bit pulse so a five and six bit pulse could be difficult to distinguish from each other.

5 Installing Code into Existing Program

5.1 Overview

Integrating this code into an existing program is relatively simple. The only added code will be initializing timer capture, integrating timer capture and the measure_baud function into the standard interrupt service routine, and creating the match_baud function as well as calling it from the main program.

5.2 Initialize Timer Capture

As mentioned in Section 3.2 Timer Capture Registers, there are a few lines of code that must be implemented in order to capture edges of the signal. Below is the main LLC code with an integrated initialization of the timer capture. Added code is in green and code already inside the LLC code is in black.

```
void init_TCAP(void)
{
      TimerRegs.T24CAPCTRL.bit.CAP_SEL = 1;          //input signal comes
from SCI_RX0 pin
      TimerRegs.T24CAPCTRL.bit.EDGE = 2;             //enable capture on
falling edge
}

void main()
{
…
      //Initialize global variables.
      init_variables();
      //Initialize GPIO pins and set initial states.
      init_gpio();
      //Initialize UARTs
      init_uart0();
      init_uart1();
      //Initialize TCAP
      init_TCAP();
      //Initialize DPWMs
      init_dpwms();
…
}
```

5.3 Timer Capture and Measure Baud Function

The timer capture inside of the measure baud function is implemented into the standard interrupt service routine. Below is the standard interrupt LLC code with an integrated measure baud function into the existing code. Everything changed from the original is in green and the original code is in black.

The measure baud function is a new function to integrate into existing code. It is further explained in Section 5 Baud Rate Matching Firmware but the entire function has been integrated into the standard_interrupt.c file with the other functions used in the interrupt service routine.

```
/* standard_interrupt.c
* measure_baud is called inside of the
* handle_standard_interrupt_global_tasks functions
* it measures the pulse width of the incoming
* signal from the UART on the PFC
*/
void measure_baud(void)
{
      T24SREG = TimerRegs.T24CAPCTRL.bit.CAP_INT_FLAG;
      if ((T24SREG == 1) && (edge == 0))        //first edge detected
      {
            result = TimerRegs.T24CAPDAT.bit.CAP_DAT;       //read and clear
            edge = 1;
                        //increase edge to capture second edge
      }
      else if ((T24SREG == 1) && (edge == 1))         //second edge detected
      {
            result = TimerRegs.T24CAPDAT.bit.CAP_DAT - result;
                              //result now contains the difference between
                        edges captured
            pulse_width = result;
                              //store result into pulse_width - will be
passed into match_baud();
            edge = 0;
                        //reset edge to detect first edge again
      }
      else
      {
      }

}
```

The measure baud function is called inside of the global tasks function inside of the standard interrupt.

```
/* standard_interrupt.c
* handle_standard_interrupt_global_tasks is called in the standard interrupt
* it handles tasks such as polling the ADC and collecting the data,
* receiving UART messages and using Timer Capture
*/
void handle_standard_interrupt_global_tasks(void)
{
      //Collect ADC data
      poll_adc();
      //Average ADC readings;
      average_adc_readings();
      //Receive UART data
      uart_receive_data();
      //call pgood handler
      handle_pgood();
```

```
      //measure baud rate with timer capture
      measure_baud();
}
```

The global tasks function is called inside of the standard interrupt service routine. This is already implemented; it is shown to further explain where everything has been called.

```
/*
 * standard_interrupt.c
 * includes measure baud timer capture interrupt function and uart receive
interrupt function
 * as well as standard interrupt
 * the only function changed / used for this application is
handle_standard_interrupt_global_tasks();
 * this function includes the timer capture interrupt and receive data
interrupt
*/
#pragma INTERRUPT(standard_interrupt,IRQ)
void standard_interrupt(void)
{
      //Set a GPIO to signify the beginning of the interrupt
//    MiscAnalogRegs.GLBIOVAL.bit.DPWM2A_IO_VALUE = 1;
      //Perform general tasks.
      handle_standard_interrupt_global_tasks();

      switch (supply_state)
      {
      case STATE_IDLE:
            handle_idle_state();
            break;
      case STATE_RAMP_UP:
            handle_ramp_up_state();
            break;
      case STATE_RAMP_DOWN:
            handle_ramp_down_state();
            break;
      case STATE_REGULATED:
            handle_regulation_state();
            break;
      case STATE_VOUT_TRANSITION:
            handle_vout_transition_state();
            break;
      case STATE_LIGHT_LOAD:
            handle_light_load_state();
            break;
      case STATE_CPCC:
            handle_cpcc_state();
            break;
      case STATE_HICCUP:
            handle_hiccup_state();
            break;
      case STATE_FAULT:
            handle_fault_state();
            break;
      }
```

```
      //clear interrupt flag by a read/write to register.
      TimerRegs.T16PWM2CMPCTRL.all = 3;
      //Clear the GPIO to signify the end of the interrupt
//    MiscAnalogRegs.GLBIOVAL.bit.DPWM2A_IO_VALUE = 0;
}
```

5.4 Match Baud Function

The match baud function must be implemented in the LLC code and called in the main function in order for the baud matching to occur.

```
/*
 * match_baud.c
 *
 *      Equations used to find bit times:
 *      baud_rate = ICLK / bit_time
 *      baud_div_value = ICLK/(8*baud_rate) - 1
 *      baud_div_value = bit_time/8 - 1
 *      baud_div_value = (bit_time >> 3) - 1
 */
#include "system_defines.h"
#include "cyclone_device.h"
#include "pmbus_commands.h"
#include "pmbus.h"
#include "variables.h"
#include "function_definitions.h"
#include "software_interrupts.h"
#include "cyclone_defines.h"

#define ICLK      (15625000)

void match_baud(Uint32 bit_time)
{
      if ((bit_time >= 2950) && (bit_time <= 3620))
                  //calculated correct pulse width (+/- 10% from 4800 baud)
      {
            baud_div_value = (bit_time >> 3)-1;
            Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
            Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
            Uart1Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
            Uart1Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);

      }
      else if ((bit_time >= 5900) && (bit_time <= 7240))
      //calculated 2x pulse width
      {
            baud_div_value = ((bit_time >> 4) & 0xffff)-1;
            Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
            Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
            Uart1Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
            Uart1Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);

      }
```

```
	else if ((bit_time >= 8850) && (bit_time <= 10860))
	//calculated 3x pulse width
	{
		baud_div_value = (((bit_time >> 4) & 0xffff)*2/3)-1;
		Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
		Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
		Uart1Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
		Uart1Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);

	}
	else if ((bit_time >= 11801)&&(bit_time <= 14481))
	//calculated 4x pulse width
	{
		baud_div_value = ((bit_time >> 5) - 1);
		Uart0Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
		Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
		Uart1Regs.UARTMBAUD.bit.BAUD_DIV_M = (baud_div_value >> 8);
		Uart1Regs.UARTLBAUD.bit.BAUD_DIV_L = (baud_div_value & 0xff);
	}
	else
	{
		//don't calculate baud rate
	}

}
```

The match baud function is called in the main function inside of the infinite loop so that it continually matches the baud rate.

```
void main()
{
	for(;;)
	{
		if (erase_segment_counter > 0)
		{
			//Handle the DFlash segment erases
			erase_task();
		}
			//Run the PMBus handler
			pmbus_handler();
			//Transmit data out the UART
			uart_transmit_data();
			//Process data received from the UART
			uart_process_rx_data();
			//Match baud
			match_baud(pulse_width);

			//output the baud rate from UART1 - used for debug purposes
			char_out_1(Uart0Regs.UARTLBAUD.bit.BAUD_DIV_L);


	}
}
```

6 References

1. B. McDonald, H. Huang. “UCD3138 64 Pin Open Loop Board”. Texas Instruments. August 2012.
2. UCD3138 Monitoring and Communications Programmer’s Manual. (Literature Number: SLU996A) Link: http://www.ti.com/lit/pdf/sluu996
3. S. Rajagopalan. Example Code.

Note: Not all Manuals may be up to date

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265