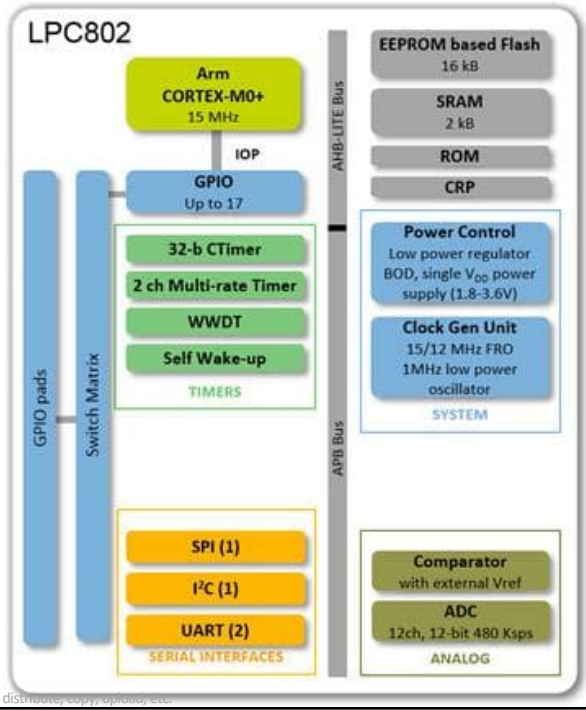
Lab 10: Serial Protocols UART

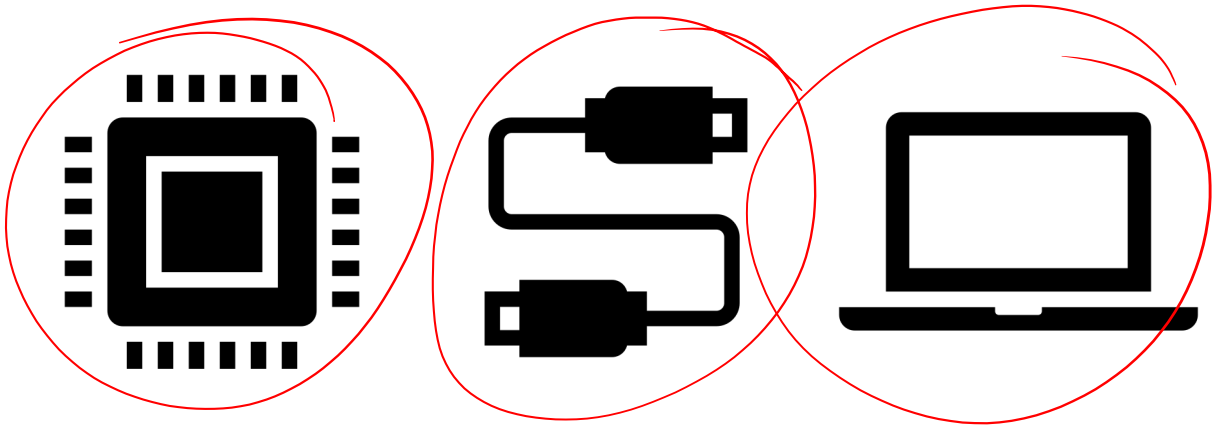
James Andrew Smith, PhD, PEng

1

Overview

- What is the UART?
  - … and USART…
- Why do we want to use it?
- Example usage
  - Serial transfers from the micro to a computer
  - Between devices: RS-485





What is serial communication?

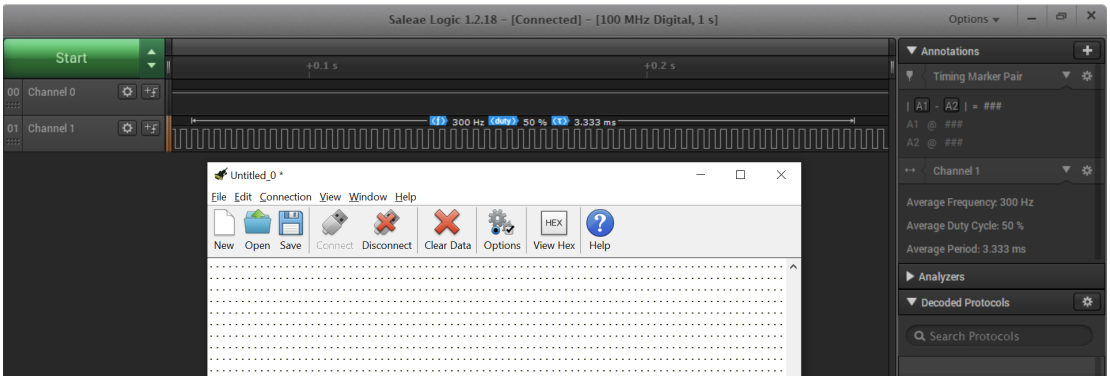
…

Serial communication

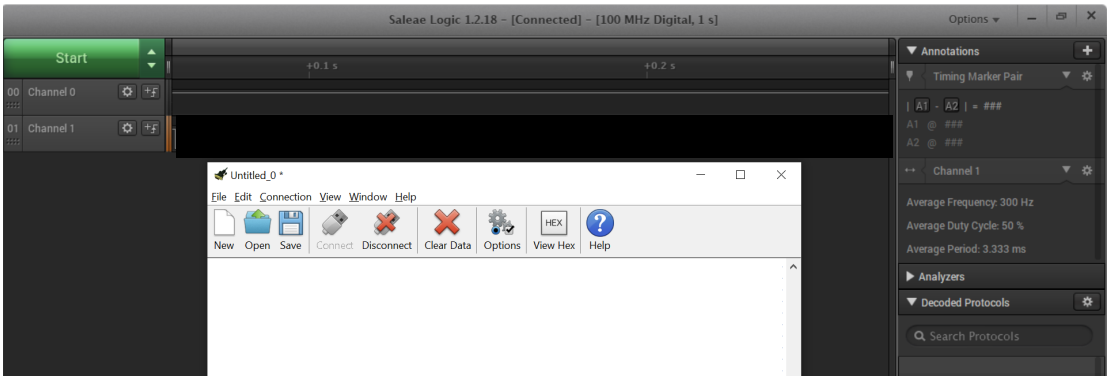
- Send a series of bits over a wire to another device.
- Each bit means something
  - If we know context
  - If we know the sequence
- The Matrix's digital rain
  - From top to bottom
  - Shows multiple serial streams

https://commons.wikimedia.org/wiki/File:Digital_rain_animation_medium_letters_shine.gif

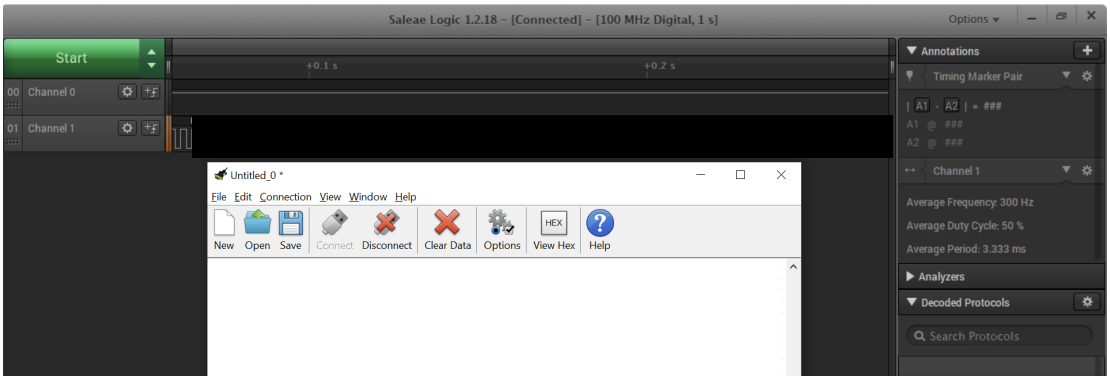
Simplest Serial Data Stream: PWM



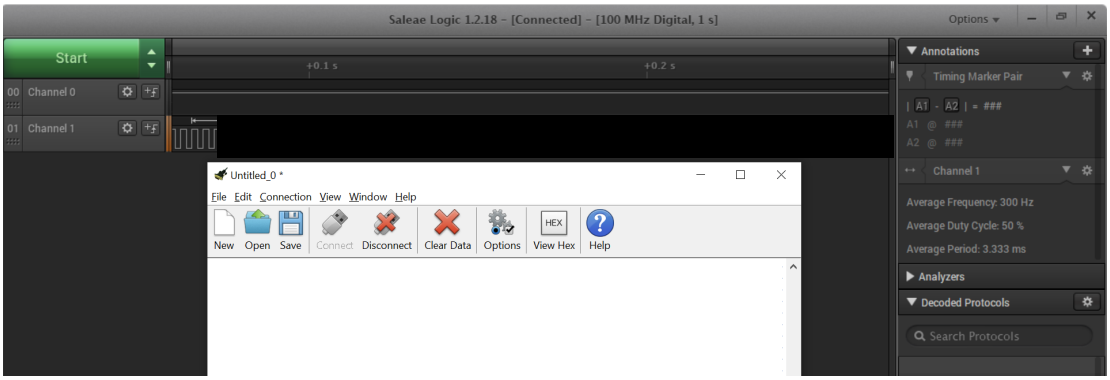
Simplest Serial Data Stream: PWM



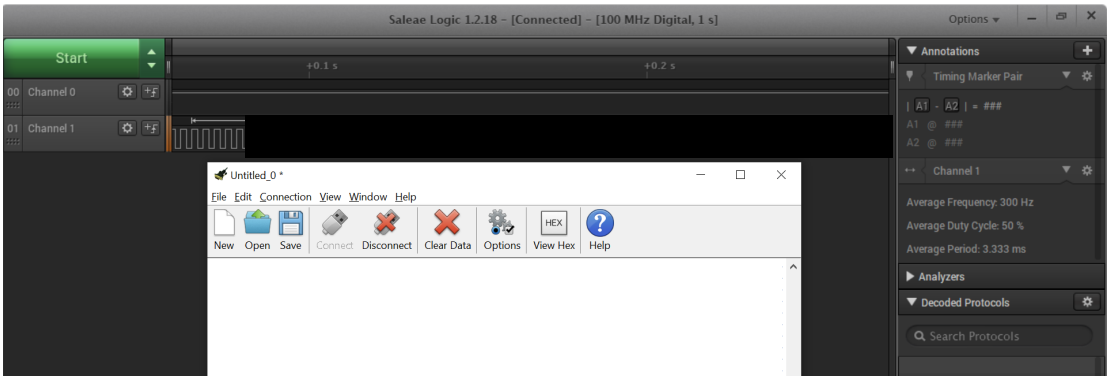
Simplest Serial Data Stream: PWM



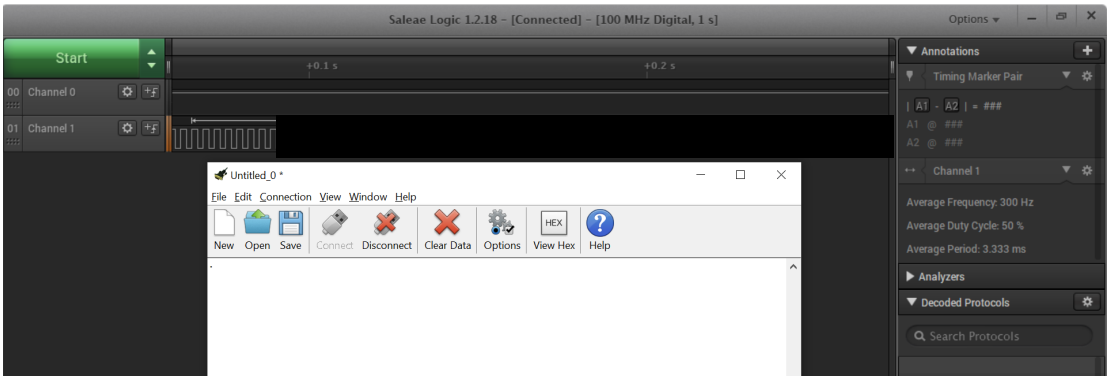
Simplest Serial Data Stream: PWM



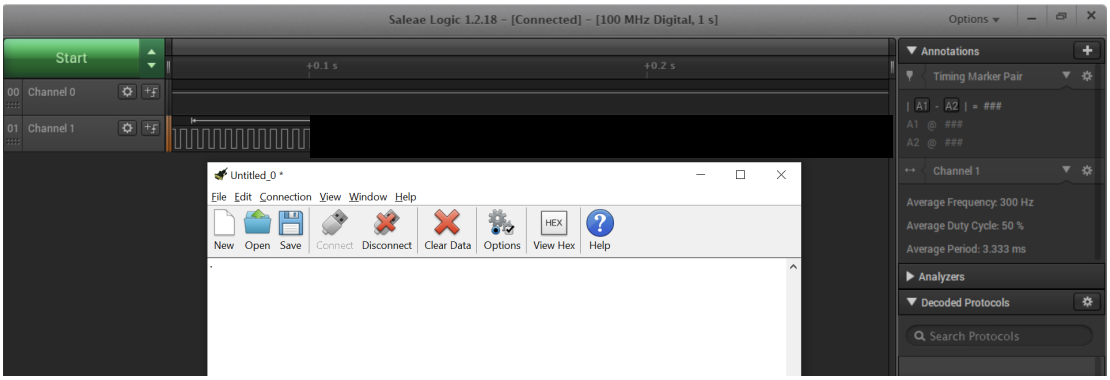
Simplest Serial Data Stream: PWM



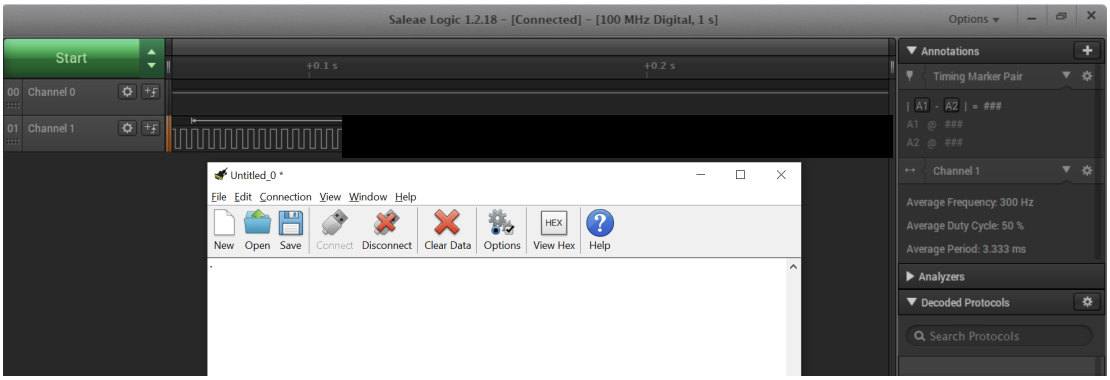
Simplest Serial Data Stream: PWM



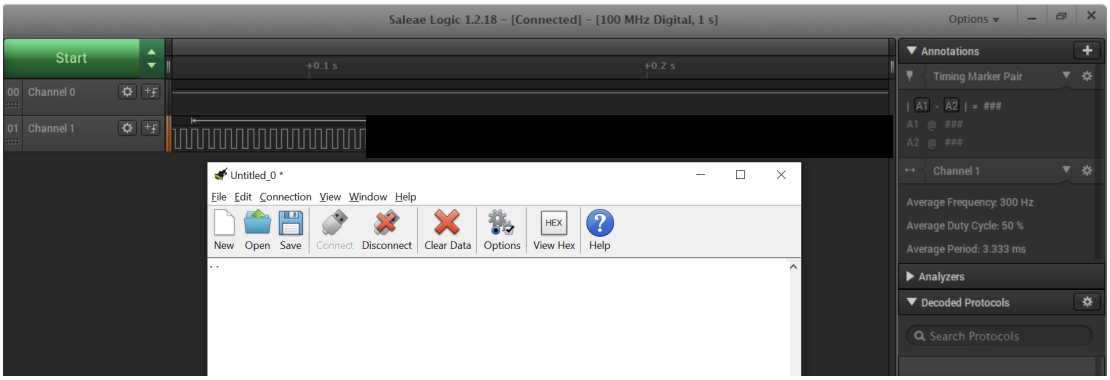
Simplest Serial Data Stream: PWM



Simplest Serial Data Stream: PWM

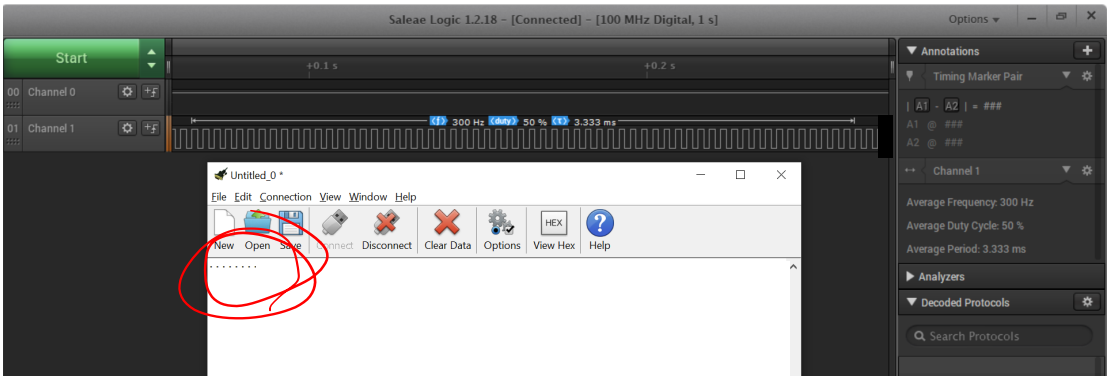


Simplest Serial Data Stream: PWM

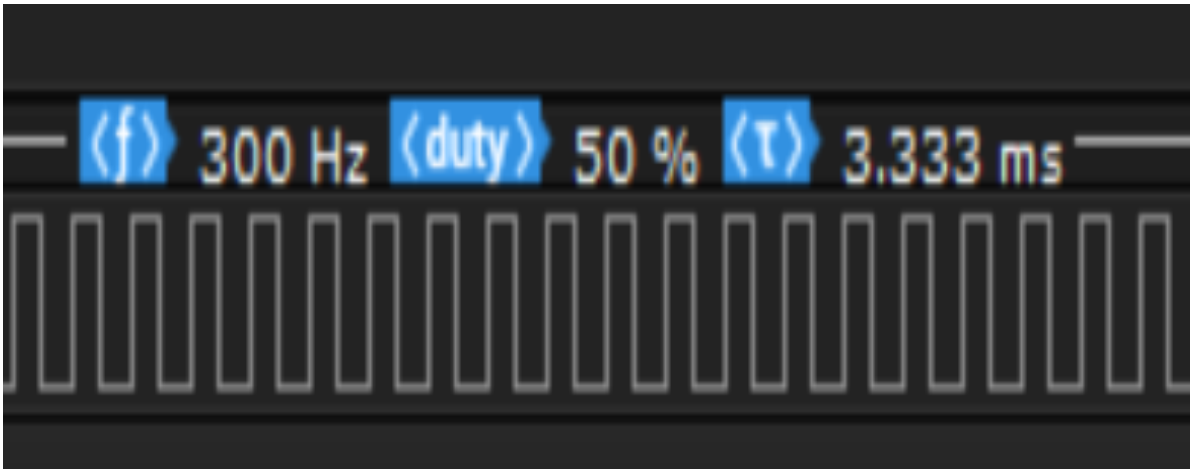


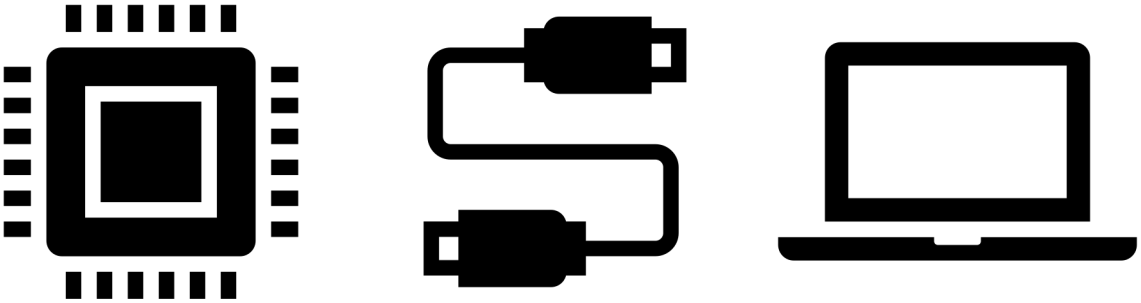
- Keep going …

Simplest Serial Data Stream: PWM



Simplest Serial Data Stream: PWM





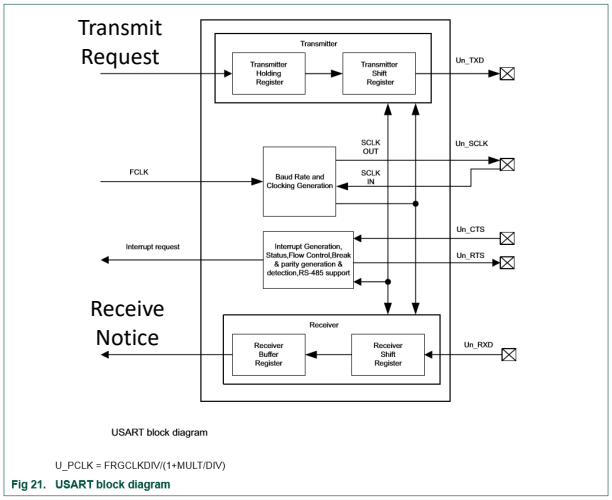
What is a UART?

Universal Asynchronous Receiver and Transmitter

The UART (aka USART)

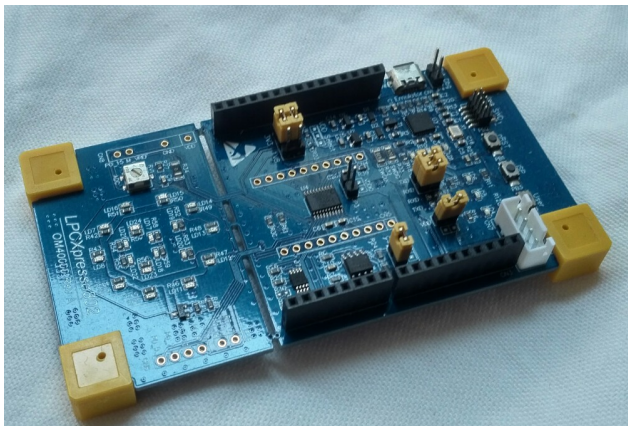
The U(S)ART

- Optional synchronous mode
- 7-9 bit packets
- 1 or 2 stop bits
- Parity Checking
- RS485 support
- Loopback mode for testing



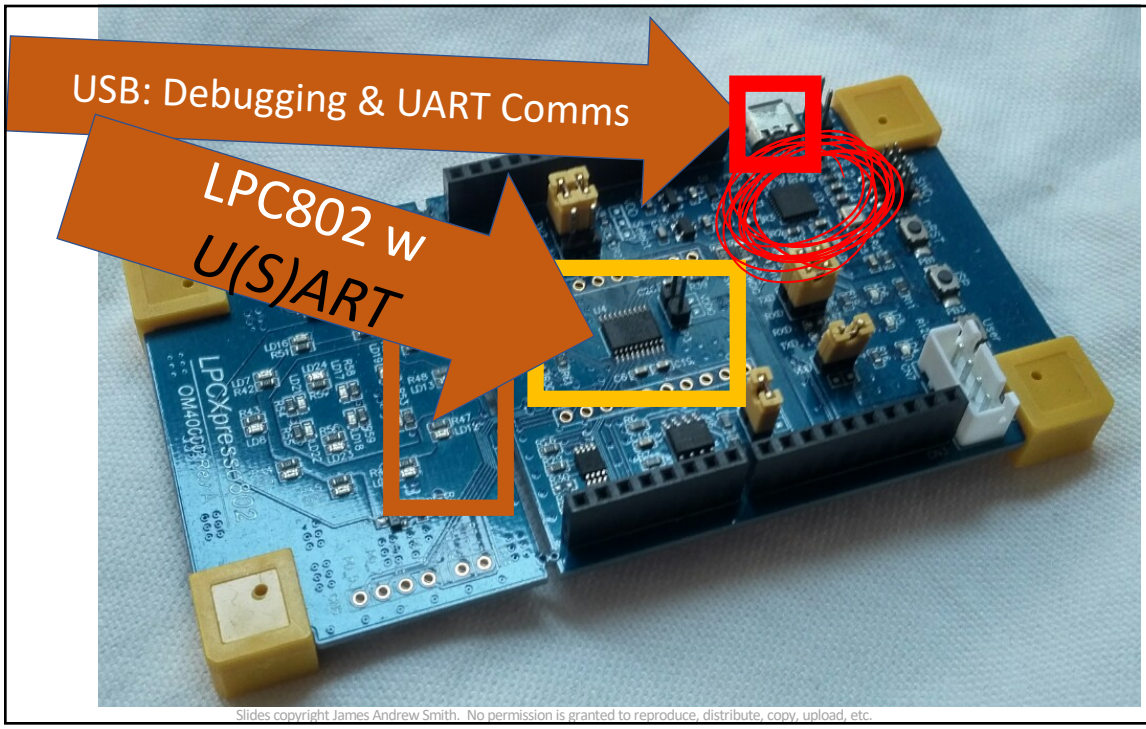
User Manual: UM11045

Note typos in user manual: No DMA.

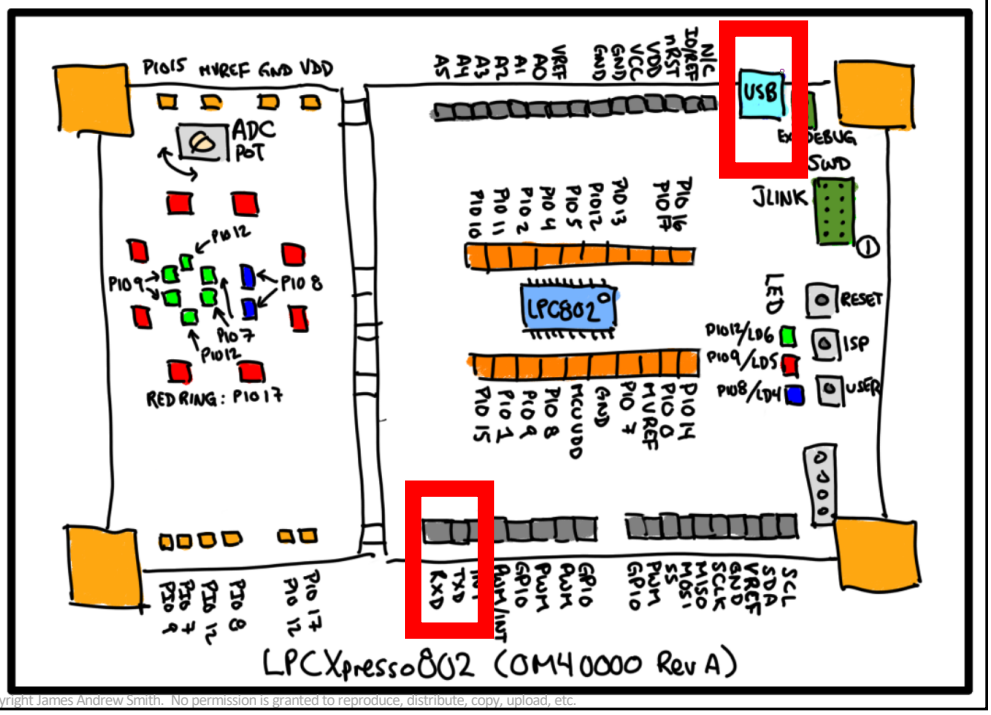


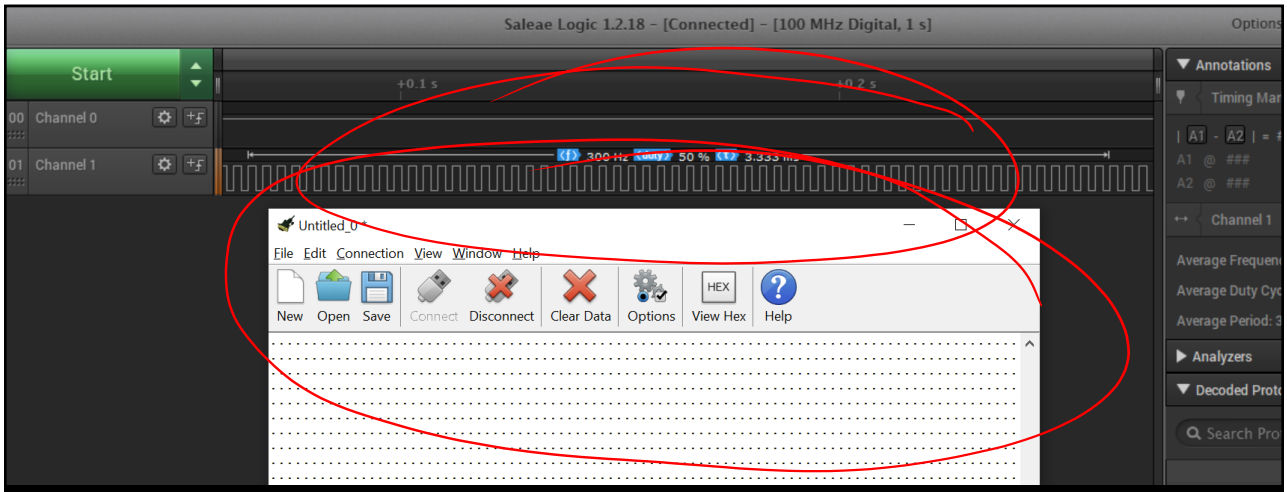
Send messages to your PC via the UART

…



The UART0 via





PWM to a PC Terminal

Simplest "Dumb" Comms

Setup

```
// ch8_pwm_txd_v1_working.c
// set up CoolTerm to listen on a USB serial port at 300 Baud.  You'll see a serial stream
show up.
// We're using GPIO pin PIO0_4 to transmit via the onboard debugger.
// ---------------------------------------------------------------------------------

#define WKT_FREQ 1000000    // Use if the WKT is clocked by the LPOSC
//#define WKT_FREQ 937500   // Use if the WKT is clocked by the 15 MHz FRO,
// via the div-by-16 WKT divider
#define WKT_RELOAD 1667  // Reload value for the WKT down counter
// 1000000 means 2 Hz LED freq (1 Hz IRQ)
// 1667 means 300 Hz GPIO toggle (600 Hz IRQ)
#define WKT_INT_FREQ (WKT_FREQ/WKT_RELOAD)  // Interrupt frequency of the WKT.
#define TXD_OUTPUT(4)// PIO0_4 is connected to the debugger's UART receive pin.
#include "LPC802.h"

// prototypes
void WKT_Config();
```


Main function (Part 1 of 1)

```
// main routine
//
int main(void) {
  // WKT: Wake Timer configuration
  WKT_Config();

  // do nothing in the main loop.
  while(1) {
  asm("NOP");
  } // end of loop.

} // end of main
```

WKT function (part 1 of 3)

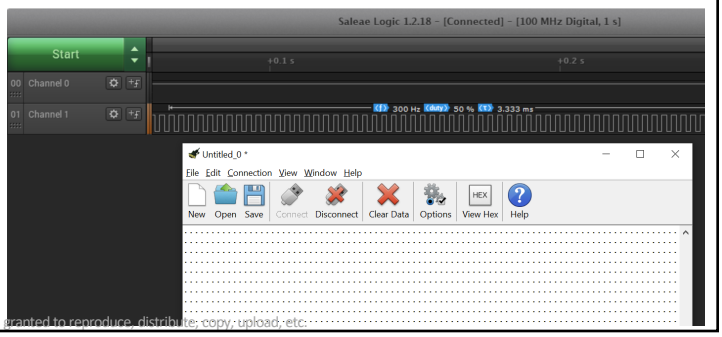
```
// Function name: WKT_Config
// Description:   Initialize a GPIO output pin and the WKT, then start it.
// Source: Ch. 18 of the User Manual (UM11045)
void WKT_Config() {

// -------------------------------------------------------
// Step 1.  Make the PIO0_4 line (TXD) an output.
SYSCON->SYSAHBCLKCTRL0 |= (SYSCON_SYSAHBCLKCTRL0_GPIO0_MASK);
// GPIO to Output.
GPIO->DIRSET[0] = (1UL<<TXD_OUTPUT);
// Turn on the LED.
GPIO->CLR[0] = (1UL<<TXD_OUTPUT);

// -------------------------------------------------------
// Step 2: turn on the Wake-up Timer (WKT) enabling clock.
SYSCON->SYSAHBCLKCTRL0 |= (SYSCON_SYSAHBCLKCTRL0_WKT_MASK);

// -------------------------------------------------------
// Step 3: disable the vector lookup for WKT.
NVIC_DisableIRQ(WKT_IRQn); // turn off the WKT interrupt.
```


WKT function (part 3 of 3)

```
// -------------------------------------------------------
// Step 7: Load the timer
// -------------------------------------------------------
// Select the LPOSC as the WKT clock source (0b0001)
// Bit 0 (CLKSEL) to 1 for Low Power Clock
// Bit 1 (ALARMFLAG) read flag.  If 1, an interrupt has happened. if 0, then no timeout.
// Bit 2 (CLEARCTR).  Set to 1 to clear the counter.
// Bit 3 (SEL_EXTCLK).  Set to 0 for internal clock source.
WKT->CTRL = WKT_CTRL_CLKSEL_MASK; // (Choose Low Power Clock using Bit 0)

// load the timer count-down value. (The "Timeout value")
WKT->COUNT = WKT_RELOAD;     // Start the WKT, counts down WKT_RELOAD clocks then interrupts

// Enable the IRQ
NVIC_EnableIRQ(WKT_IRQn);    // Enable the WKT interrupt in the NVIC
}
```

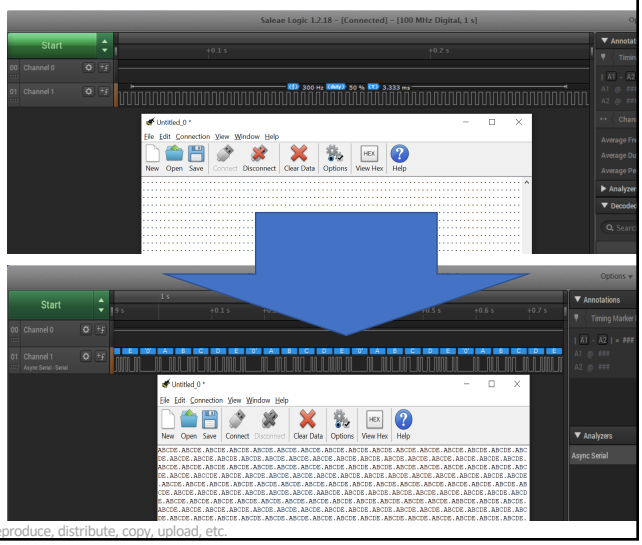
WKT ISR

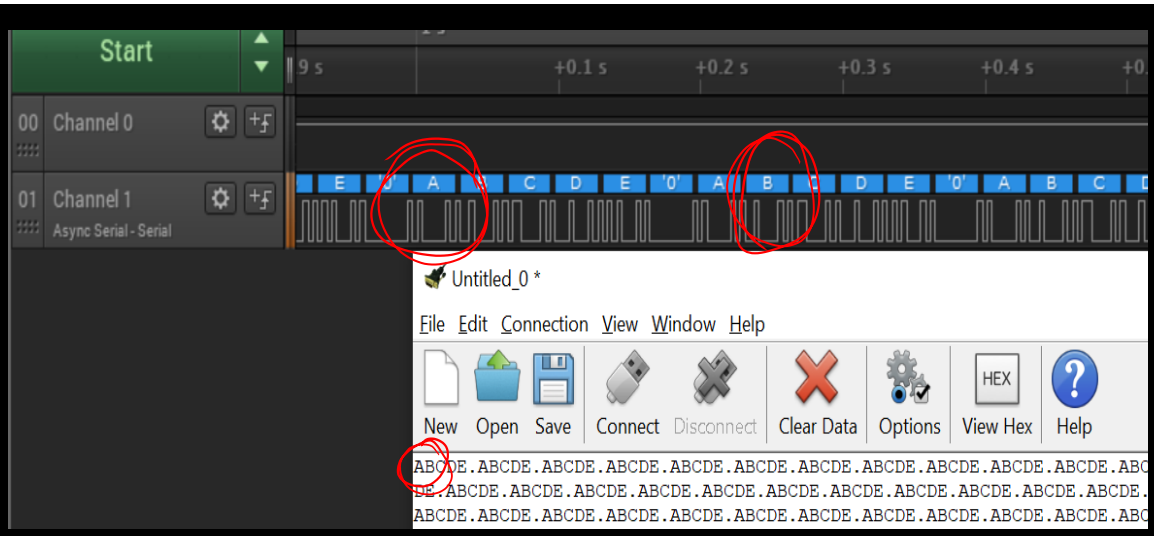
```
// Function name: WKT_IRQHandler (interrupt service routine)
// Description:   WKT interrupt service routine.
//                Toggles a GPIO line (PIO0_4) & restarts the WKT.
// Parameters:    None
// Returns:       Void
void WKT_IRQHandler(void) {
  WKT->CTRL |= WKT_CTRL_ALARMFLAG_MASK; // Clear the interrupt flag
  WKT->COUNT = WKT_RELOAD;              // Restart the WKT

  // Toggle the TXD line (PIO0_4)
  GPIO->NOT[0] = (1UL<<TXD_OUTPUT);

  return;  // return nothing.
}
```



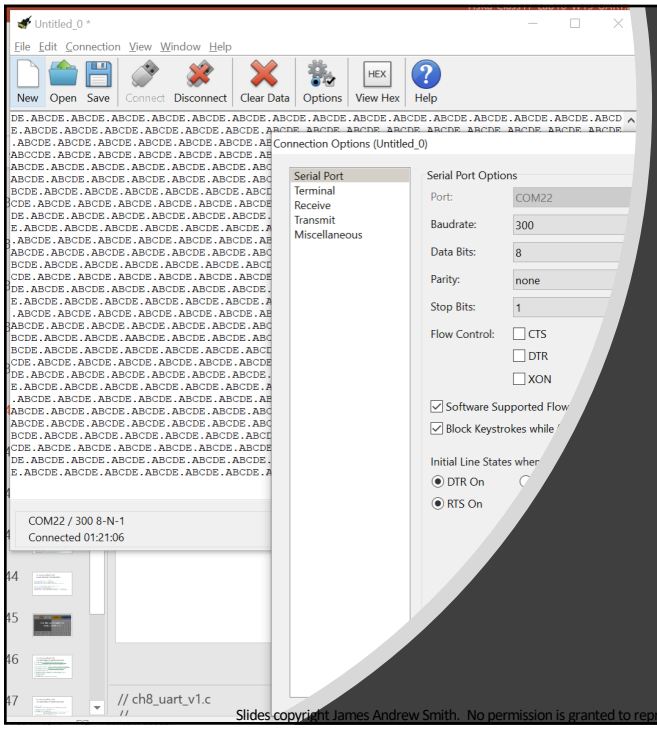
Let's make meaningful signals now

- PWM shows us that it works.
- But the signal is not very meaningful.
- Use the USART module to transmit signals





USART tx'ing ABCDE at 300 Baud | Typical Serial Comms



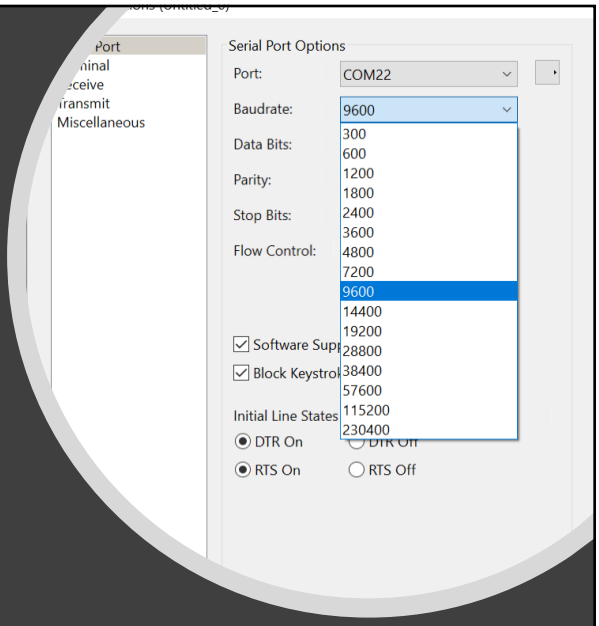
Example: ABCDE at 300 bps on USART

Set the transmission frequency

- Traditionally, UART devices
  - Don't have a clock / timing / synchronizing wire
  - Devices are not synchronized
  - Rely on an assumption about frequency
  - Both the transmitter and receiver need to agree about the frequency
  - A priori
- New : the "S" in USART
  - Synchronize
  - Also "auto-baud" detection
  - (we won't cover this)

Setting Transmission Frequency

- Free programs like CoolTerm provide example values
- Frequencies are measured in
  - Bits per second (bps or "Baud")
  - Invert time to transmit a single complete bit (high or low)
- Only certain frequencies are commonly used
  - 300 bits per second (baud)
    - Human readable on your screen
  - 9600 bps (baud)
    - Common for faxes
  - 115200 bps (baud)
    - Fastest internet in 2000s
  - Higher frequencies possible



Setting Transmission Frequency on the LPC802

- Throttle down a high-frequency (MHz) clock
- Minimize error b/w desired & actual baud rate
  - Usually OK except @ high freq. in some applications
- Five variables
  - Source clock (e.g. 12 MHz)
  - MULT (multiplier)
  - DIV (divider)
  - BRG (baud rate generator)
  - OSR (over sampling register)
- Over-dimensioned problem needs restrictions
  - OSR should be 16 (default; 0xFF + 1 added internally)
  - DIV can only be 256 (default)
  - MULT is 1..255
  - BRG is 1..65535
  - Source clock: can be more than just FRO
- Still large search space…

Equation for setting baud rate

$$baud = \frac{freq_{input}}{(1 + {}^{MULT}/_{DIV})} \frac{1}{(1 + OSR)} \frac{1}{(1 + BRG)}$$

MATLAB search while minimizing error:

```
actual_baud = (input_freq/(1+(mult_var(i)/div_var)))*(1/(1+osr_var))*(1/(1+brg_var(j)));
```

Matlab script search

```
input_freq = 12000000;     % 12 MHz clock.  Could be 15MHz and 9MHz, etc.
mult_var = [1:1:255];      % 1.. 255
div_var = 256;             % only 256 allowed
osr_var = 15;              % will only use 15 for now; divisor is OSR+1
brg_var = [1:1:65535];     % 1..0xFFFF; divisor is BRG+1.
baud_rates = [300 9600 14400 57600 115200];
graph_col = 1;

for k=1:(length(baud_rates))
  desired_baud_rate = baud_rates(k);
  previous_error_baud = 1000000000;
  current_error_baud = 0;
  best_mult = -1;
  best_brg = -1;
  best_baud = -1;

  for i=1:(length(mult_var))
    for j=1:(length(brg_var))
      actual_baud = (input_freq/(1+(mult_var(i)/div_var)))*(1/(1+osr_var))*(1/(1+brg_var(j)));
      current_error_baud = abs(baud_rates(k)-actual_baud);
% debug for graphing
      result(1,graph_col) = actual_baud;
      result(2,graph_col) = current_error_baud;
      result(3,graph_col) = mult_var(i);
      result(4,graph_col) = brg_var(j);
      graph_col = graph_col + 1;

      % is this the closest match?
      if (current_error_baud<=previous_error_baud)
        best_mult = mult_var(i);
        best_brg = brg_var(j);
        best_baud = actual_baud;
        previous_error_baud = current_error_baud;  % update error with smallest to date.
      end
    end
  end
% Print result
best_mult
best_brg
best_baud
previous_error_baud

% reset max error
previous_error_baud = 1000000000;
end
```


Register settings for 3 FRO values

Table 1 Settings at 9 MHz clock to achieve desired baud rates (Matlab script)

Baud rate	Baud Error	MULT	BRG	OSR reg value[1]	Main Freq
300	0	244	959	0xF	9 MHz
9600	0	244	29	0xF	9 MHz
14400	0	244	19	0xF	9 MHz
57600	0	244	4	0xF	9 MHz
115200	92.0863	161	2	0xF	9 MHz

Table 2 Settings at 12 MHz clock to achieve desired baud rates (Matlab script)

Baud rate	Baud Error	MULT	BRG	OSR reg value	Main Freq
300	0	244	1279	0xF	12 MHz
9600	0	244	39	0xF	12 MHz
14400	1.4401	148	32	0xF	12 MHz
57600	5.7606	47	10	0xF	12 MHz
115200	92.0863	22	5	0xF	12 MHz

Table 3 Settings at 15 MHz clock to achieve desired baud rates (Matlab script)

Baud rate	Baud Error	MULT	BRG	OSR reg value	Main Freq
300	0	244	1599	0xF	15 MHz
9600	0	244	49	0xF	15 MHz
14400	1.1519	207	35	0xF	15 MHz
57600	4.6076	207	8	0xF	15 MHz
115200	73.7752	91	5	0xF	15 MHz

Setup

```
// ch8_uart_v1.c
//
// To configure the USART, refer to section 13.3 of the User Manual.
//
// may 14, 2019

#include <LPC802.h>
#include "clock_config.h"  // for BOARD_BootClockFRO30M(), etc.
```


Main function (Part 1 of 9)

```
void main(void)
{
uint32_t i = 0;
uint8_t temp =0;
uint8_t my_string[6] = {'A','B','C','D','E','\0'};

    // disable interrupts
    __disable_irq();     // turn off globally
```

10

Set up main clock…

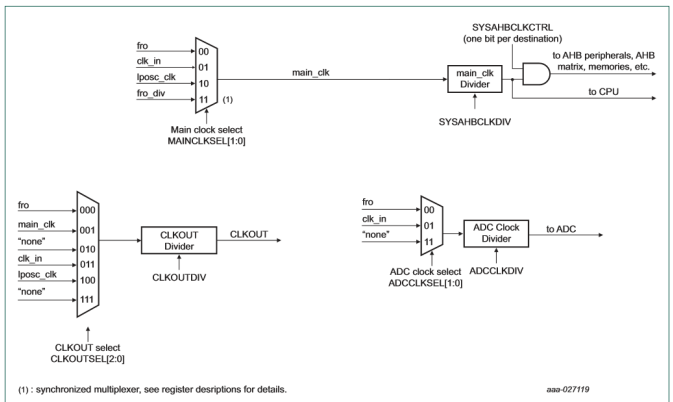


Fig 6. LPC802 clock generation

Main function (Part 2 of 9)
Set up main clock (12 MHz)

```
// ---------------------- Begin Core Clock Select  ----------------------
// Specify that we will use the Free-Running Oscillator
// Set the main clock to be the FRO
// 0x0 is FRO; 0x1 is external clock ; 0x2 is Low pwr osc.; 0x3 is FRO DIV
// Place in bits 1:0 of MAINCLKSEL.
SYSCON->MAINCLKSEL = (0x0<<SYSCON_MAINCLKSEL_SEL_SHIFT);

// Update the Main Clock
// Step 1. write 0 to bit 0 of this register
// Step 2. write 1 to bit 0 this register
SYSCON->MAINCLKUEN &= ~(0x1);// step 1. (Sec. 6.6.4 of manual)
SYSCON->MAINCLKUEN |= 0x1;// step 2. (Sec. 6.6.4 of manual)

// Set the FRO frequency (clock_config.h in SDK)
//
// For FRO at 9MHz: BOARD_BootClockFRO18M();
// 12MHz: BOARD_BootClockFRO24M();
// 15MHz: BOARD_BootClockFRO30M();
// See: Section 7.4 User Manual
// This is more complete than just using set_fro_frequency(24000); for 12 MHz
BOARD_BootClockFRO24M();  // 30M, 24M or 18M for 15MHz, 12MHz or 9MHz
// ------------ End of Core Clock Select ------------
```

Main function (Part 3 of 9)

Turn on the USART (#0) and reset it

```
// -------------- Step 1: Turn on the USART0 ---------------------
// Bit is 14 (USART0).  Set to 1 to turn on USART0.
// Optionally, we could also turn on USART1 with bit 15
SYSCON->SYSAHBCLKCTRL0 |= (SYSCON_SYSAHBCLKCTRL0_UART0_MASK);  // Set bit 14.

// -------------- Step 2: Reset the USART0 ---------------------
//  Set bit 14 to 0: assert (i.e. "make") the USART0 reset
//  Set bit 14 to 1: remove the USART0 reset
SYSCON->PRESETCTRL0 &= ~(SYSCON_PRESETCTRL0_UART0_RST_N_MASK);    // Reset USART0
SYSCON->PRESETCTRL0 |= (SYSCON_PRESETCTRL0_UART0_RST_N_MASK);     // remove the reset.
```


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1				Fixed Func	Extra Func	Package Pi	Board Hea								
2	Name	Register	Pin Enable 0	-	-	-		PinAssign 0				PinAssign 1			
3			SWM0->PINENABLE0					SWM0->PINASSIGN0				SWM0->PINASSIGN1			
4			Fixed Func	Function	-	M001JDH	-	USART 0					USART 1		USART 0
5			Bits 21:0		-	TSSOP20	-	Bits 31:24	Bits 23:16	Bits 15:8	Bits 7:0	Bits 31:24	Bits 23:16	Bits 15:8	Bits 7:0
6			-	-	-		-	U0 CTS	U0 RTS	U0 RXD	U0 TXD	U1 SCLK	U1 RXD	U1 TXD	U0 SCLK
7			-	-	-		-	SWM_PINASSIGN0_U0_CTS_I_MASK	SWM_PINASSIGN0_U0_RTS_O_MASK	SWM_PINASSIGN0_U0_RXD_I_MASK	SWM_PINASSIGN0_U0_TXD_O_MASK	SWM_PINASSIGN1_U1_SCLK_IO_MASK	SWM_PINASSIGN1_U1_RXD_I_MASK	SWM_PINASSIGN1_U1_TXD_O_MASK	SWM_PINASSIGN1_U0_SCLK_IO_MAS
8	GPIO 0	PIO0_0	Bit 0	ACMP_I1	TDO	Pin 19	CPU_TX (C	yes	yes	yes	yes	yes	yes	yes	yes
9	GPIO 1	PIO0_1	Bit 1	ACMP_I2	ACMP_I2 /	Pin 12	MISO (CN	yes	yes	yes	yes	yes	yes	yes	yes
10	GPIO 2	PIO0_2	Bit 5	SWDIO	TMS	Pin 8	-	yes	yes	yes	yes	yes	yes	yes	yes
11	GPIO 3	PIO0_3	Bit 4	SWCLK	TCK	Pin 7	-	yes	yes	yes	yes	yes	yes	yes	yes
12	GPIO 4	PIO0_4	Bit 21	ADC_11	TRST*	Pin 6	CPU_RX (C	yes	yes	yes	yes	yes	yes	yes	yes
13	GPIO 5	PIO0_5	Bit 6	RESETN	-	Pin 5	RESET (CN	yes	yes	yes	yes	yes	yes	yes	yes

Use the Switch Matrix to assign USART pins

Main function (Part 4 of 9)
Use Switch Matrix to connect the USART

```
// ------ Step 3: Connect PIO0_4 & _0 to USART0 via Switch Matrix -------------
// The '802 is designed to have a USART route to a PC over USB by taking
// advantage of the onboard debugger and connecting to it first.  The debugger
// then routes the TXD and RXD UART lines in a way that make them appear as a
// virtual serial port to the PC over USB.
//
// Out-of-the-box, the schematic (pg 3) shows us that the '802 PIO0_4 is supposed to be
// TXD and PIO0_0 is supposed to be RXD, but only if the JP5 and JP6 jumpers are installed.
// Check the board... they are.  Alternatively, if you remove JP5 and JP6 jumpers
// and solder-bridge (or install 0 Ohm resistors) R37 and R38 you can use
// PIO0_8 as RXD and PIO0_9 as TXD.
//
// Switch Matrix PINASSIGN0 is responsible for RXD (Bits 15:8) and TXD (Bits 7:0)
// Some other Pin assignments for USART0 can be found in PINASSIGN1, along with
// mux settings for USART1.

// enable switch matrix
SYSCON->SYSAHBCLKCTRL0 |= (SYSCON_SYSAHBCLKCTRL0_SWM_MASK);
```


Main function (Part 5 of 9)
Use Switch Matrix to connect the USART

```
// Set switch matrix
// Clear bits 15:0
SWM0->PINASSIGN0 &= ~(SWM_PINASSIGN0_U0_TXD_O_MASK | SWM_PINASSIGN0_U0_RXD_I_MASK);

// Assign TXD and RXD ports to PINASSIGN0 bits 15:0
// TXD is PIO0_4, so put 4 into bits 7:0
// RXD is PIO0_0, so put 0 into bits 15:8
SWM0->PINASSIGN0 |= ( (0x4UL<<SWM_PINASSIGN0_U0_TXD_O_SHIFT) | // TXD is PIO0_4 into 7:0
                      (0x0UL<<SWM_PINASSIGN0_U0_RXD_I_SHIFT)); // RXD is PIO0_0

// USART0 is now set to PIO0_4 for TX and PIO0_0 for RX.

// disable the switch matrix
SYSCON->SYSAHBCLKCTRL0 &= ~(SYSCON_SYSAHBCLKCTRL0_SWM_MASK);
// ---------------- End of Switch Matrix code ------------------------------------
```

Set up UART clock

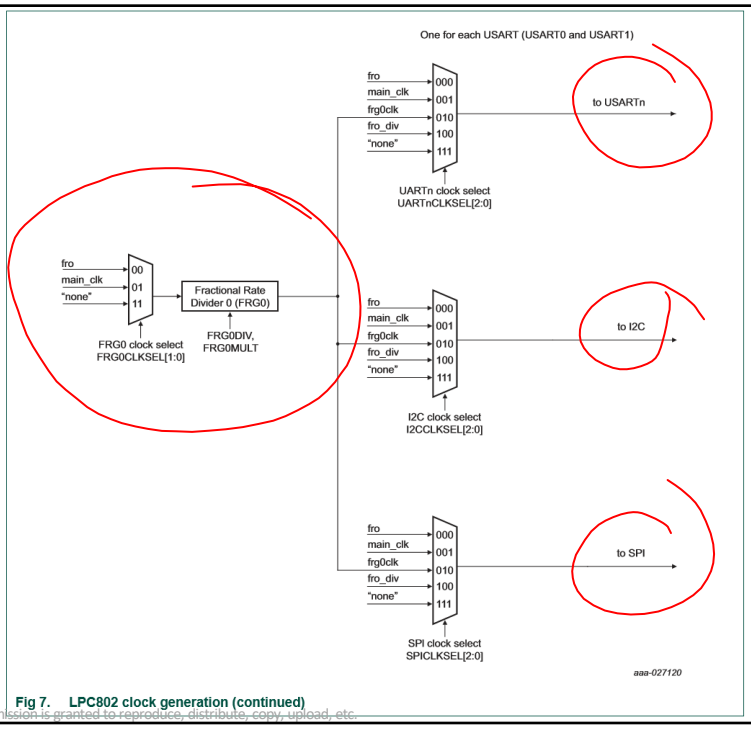


Fig 7. LPC802 clock generation (continued)

Set up UART clock

Table 2 Settings at 12 MHz clock to achieve desired baud rates (Matlab script)

Baud rate	Baud Error	MULT	BRG	OSR reg value	Main Freq
300	0	244	1279	0xF	12 MHz
14400	1.4401	148	32	0xF	12 MHz
57600	5.7606	47	10	0xF	12 MHz
115200	92.0863	22	5	0xF	12 MHz

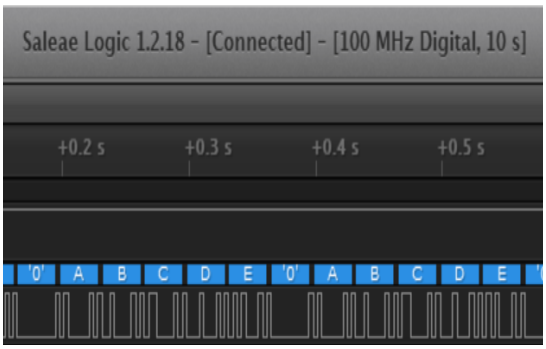
Main function (Part 6 of 9)
Set the frequency for the USART (FRG: MULT & DIV)

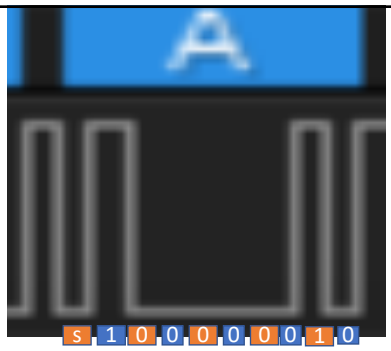
```
// -------------- Step 4: Configure USART0 Clock --------------------
// Use FRG as the input to the USART's BRG
SYSCON->UART0CLKSEL = 0x02;// FRG is 0b010

// Step 4a.  Choose clock source for clock, stage 1: FRG0
// both FRO and main clock have been set earlier.
//usually, FRO is 12 Mhz or 15 Mhz.
SYSCON->FRG[0].FRGCLKSEL = 0x00;// Choose FRO (0x0) or Main Clock (0x1)

// Step 4b. Set the fractional clock divider and multiplier pre-scaler values
// In synchronous mode FCLK = (FRGINPUTCLK)/(1+(MULT/DIV)
// or MULT = DIV*((FRGINPUTCLK/FCLK)-1)
// FRGINPUTCLK is 12 MHz and FCLK is the desired data rate and DIV is 256.
SYSCON->FRG[0].FRGDIV = 0xFF;// DIV is always 255 (0xFF); internally, it will be the programmed
value PLUS ONE.  So 255 becomes 256 in the calculation. divisor value is DIV+1 (so 255+1 =
256); according to user manual (6.6.14; Table 67)
SYSCON->FRG[0].FRGMULT = 244;// MULT is 0 to 255.  Unlike DIV, the programmed value is the
ACTUAL value used in the calculation.
```

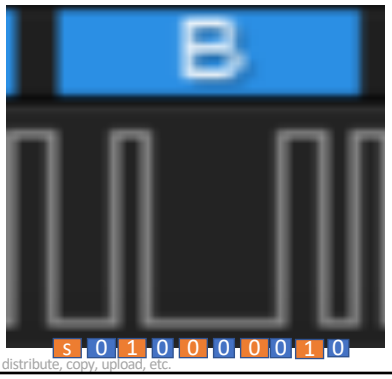

Main function (Part 7 of 9)
Set the frequency for the USART: BRG & OSR

```
// Step 4b.  Choose the BRG value
// If 0, then the output of the BRG is same freq as input freq. (divisor: BRG+1=1)
// If 1, then output of BRG is divided by 2 (divisor: BRG+1=2)
USART0->BRG = 1279;

// Oversampling within the Baud Rate Generator.
// Default is 15 (0xF ) out of reset, making it oversample OSR+1 (16) times.
// Recommend it remain 16 times over-sampling as per NXP recommendations.
// but it can be tweaked down to OSR=4 to allow 5 times oversampling.
// Cannot go below OSR=4 and cannot go above OSR=15.
USART0->OSR = 0xF;// Oversample at OSR+1 = 16 times.
```

Shape the packets…







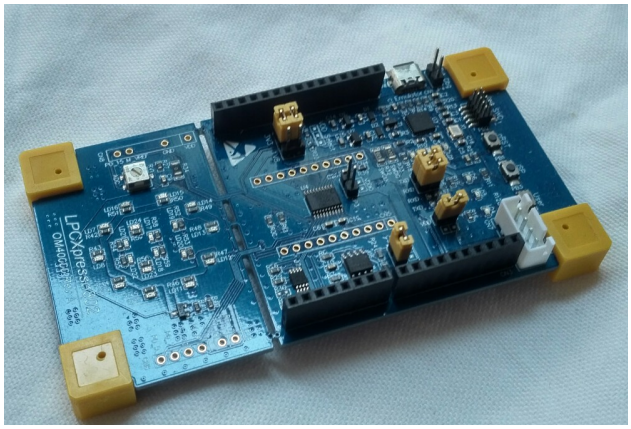
'A' = 0x41 = 0b0100 0001
'B' = 0x42 = 0b0100 0010

Main function (Part 8 of 9)
Packet length, stop bits, parity? Then Enable.

```
// USART CONFIG (only one bit to set here, really: data length)
USART0->CFG |= (      0x1<<USART_CFG_DATALEN_SHIFT |       // Data length: 8
                      0x0<<USART_CFG_PARITYSEL_SHIFT |     // Parity: none
                      0x0<<USART_CFG_STOPLEN_SHIFT);       // 1 stop bit.

// nothing to be done here.
USART0->CTL = 0;

// clear any pending flags, just in case something happened.
USART0->STAT = 0xFFFF;

// Enable the USART0
USART0->CFG |= USART_CFG_ENABLE_MASK;// set bit 1 to 1.

// -------------- Step 5: Enable interrupts  ---------------------
__enable_irq();     // turn on globally
```

Main function (Part 9 of 9)
Infinite Loop: *Send the characters, over and over*

```
while(1)
{
        // Send all characters (ABCDE)
        do
        {
                temp = my_string[i++];
                while(!((USART0->STAT) & USART_STAT_TXRDY_MASK));// wait for TX ready
                USART0->TXDAT = temp;// place in transmit buffer

        }while (temp!='\0');// keep going until you reach "null"

        asm("NOP");
        i=0;// reset counter

}
return;
} // end main function
```

Your Turn. Rewrite to send your name