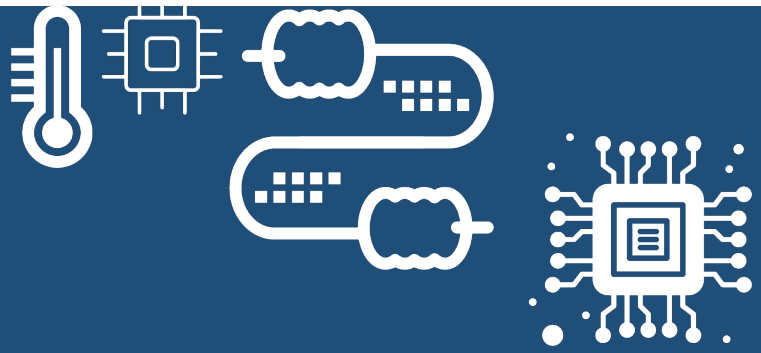


EECS 3215

Embedded Systems

Session 16: serial communications 1

Exploring i2c

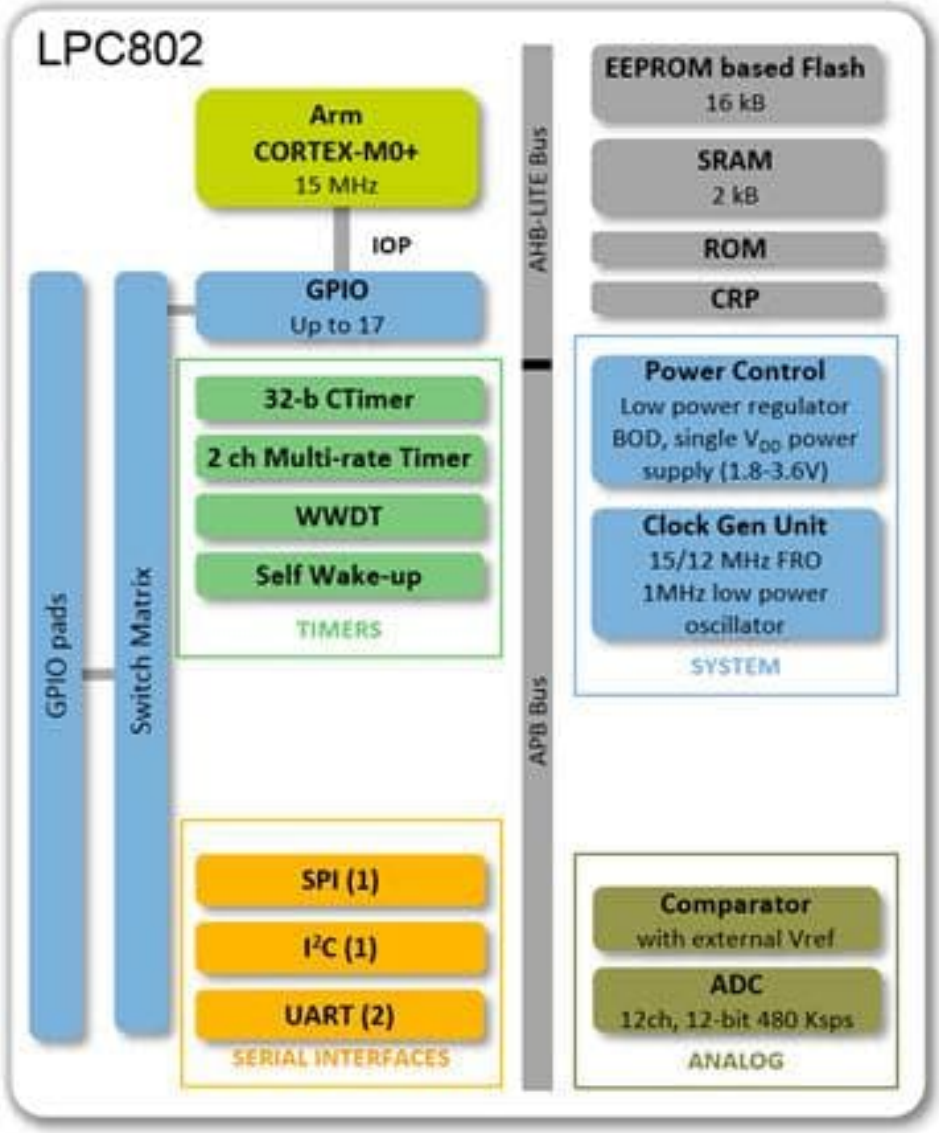
York University

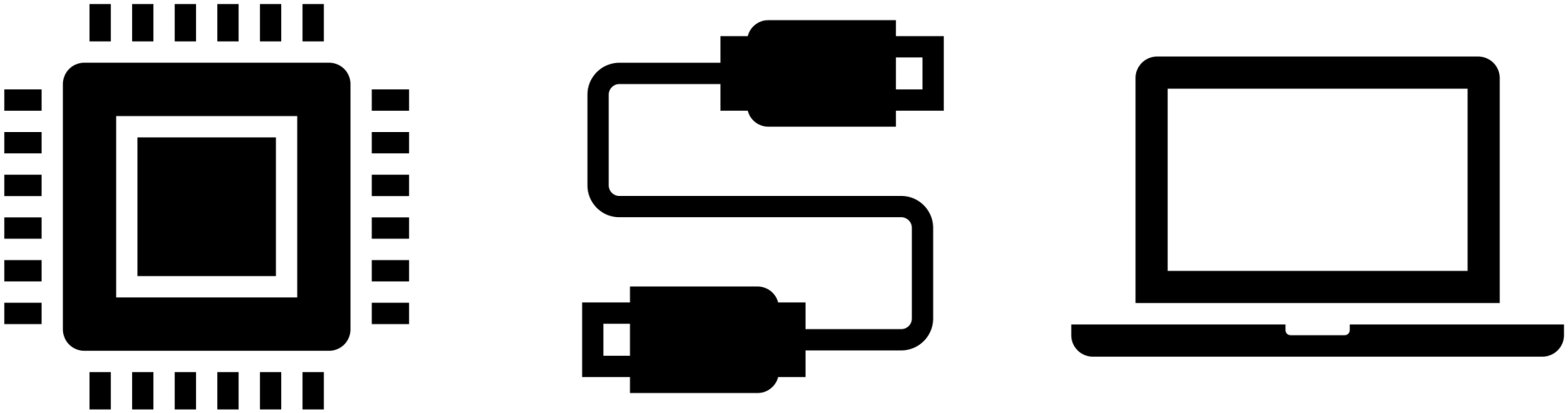
Lassonde School of Engineering

Dr. James Andrew Smith, PhD, PEng.

Overview

- What is the i2c?
- Example usage
  - Reading a temperature sensor





What is serial communication?

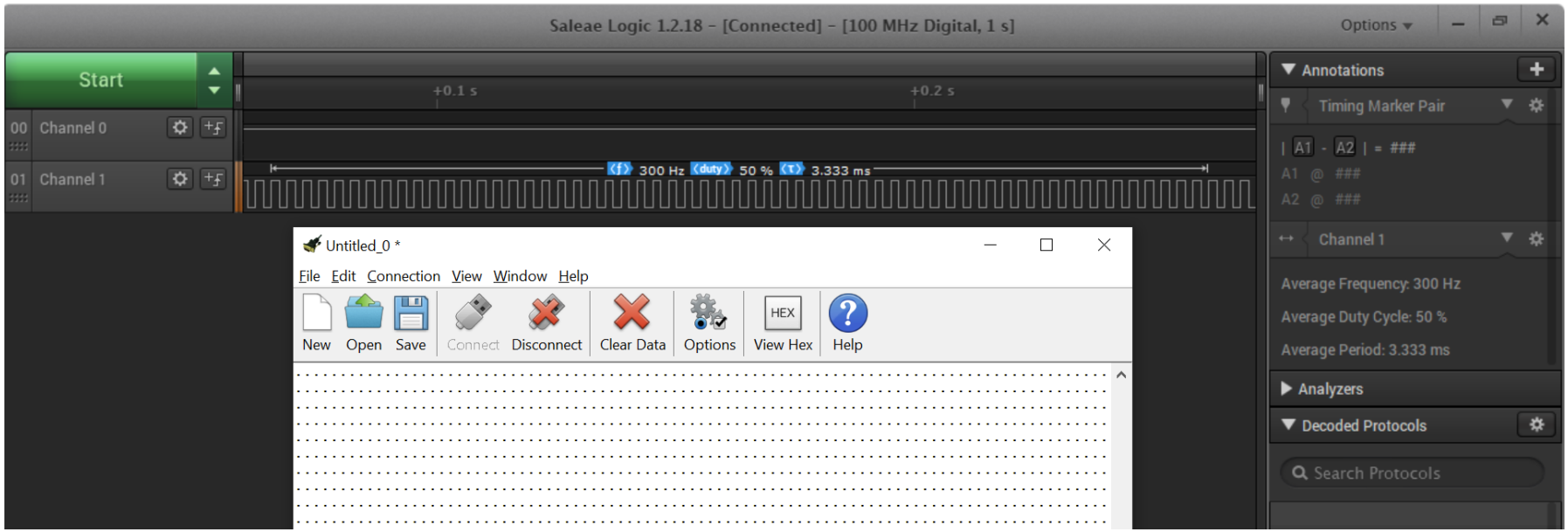
…

Serial communication

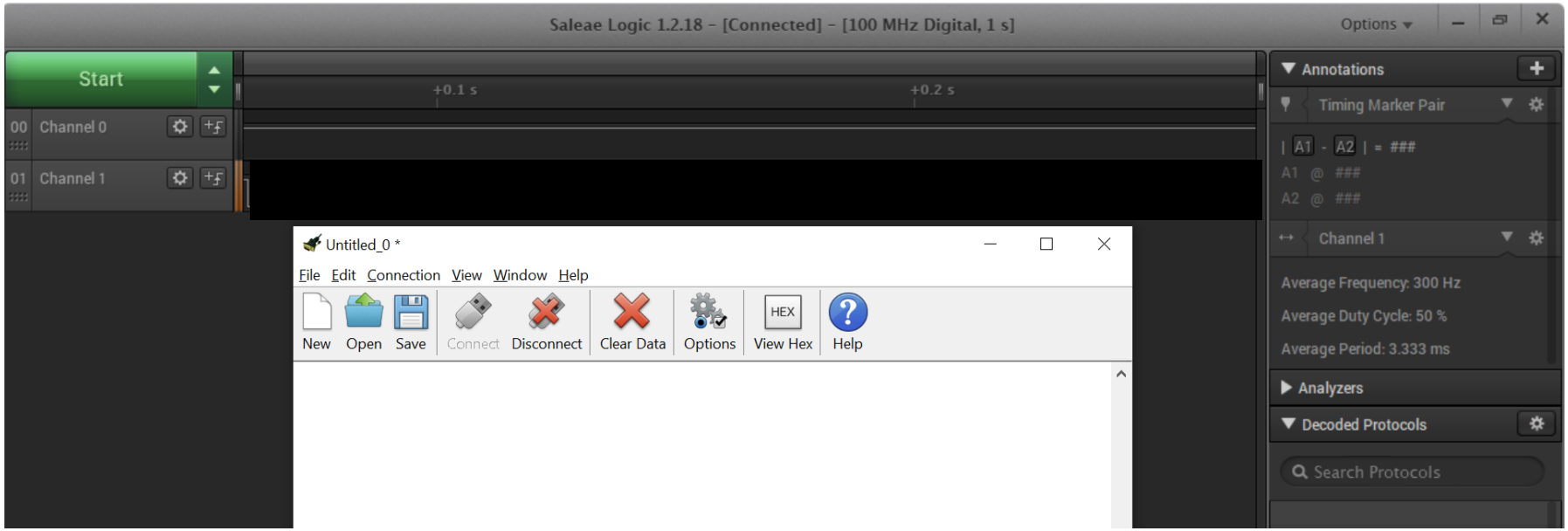
- Send a series of bits over a wire to another device.
- Each bit means something
  - If we know context
  - If we know the sequence
- The Matrix's digital rain
  - From top to bottom
  - Shows multiple serial streams

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

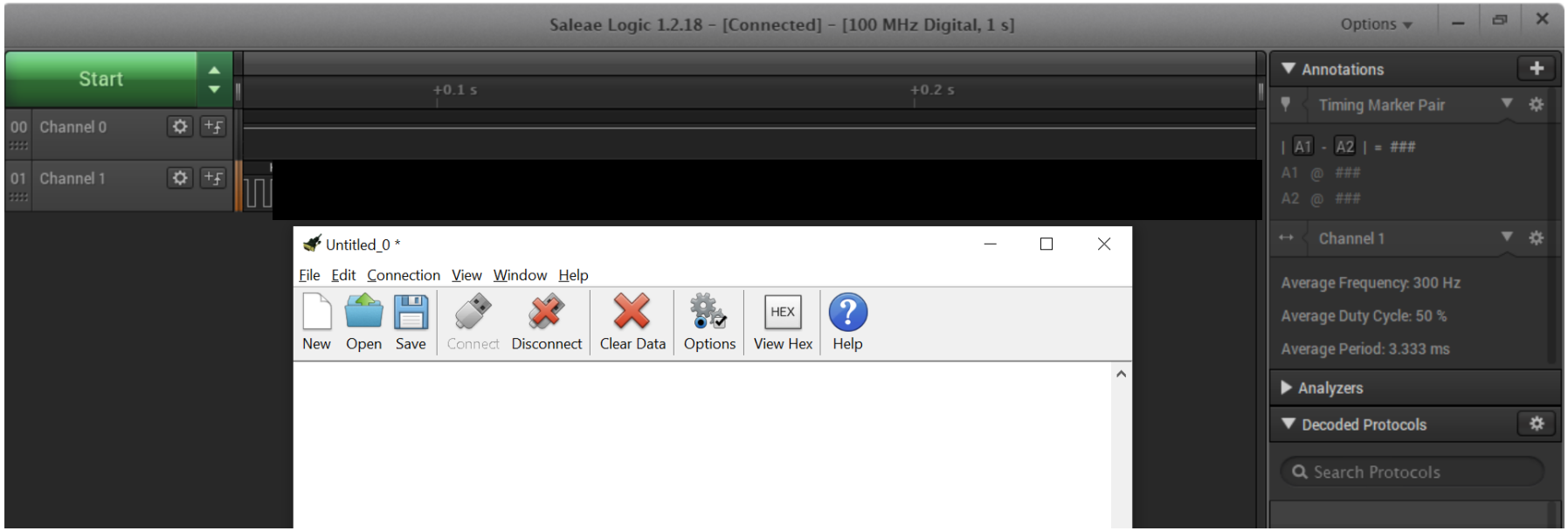
Simplest Serial Data Stream: PWM



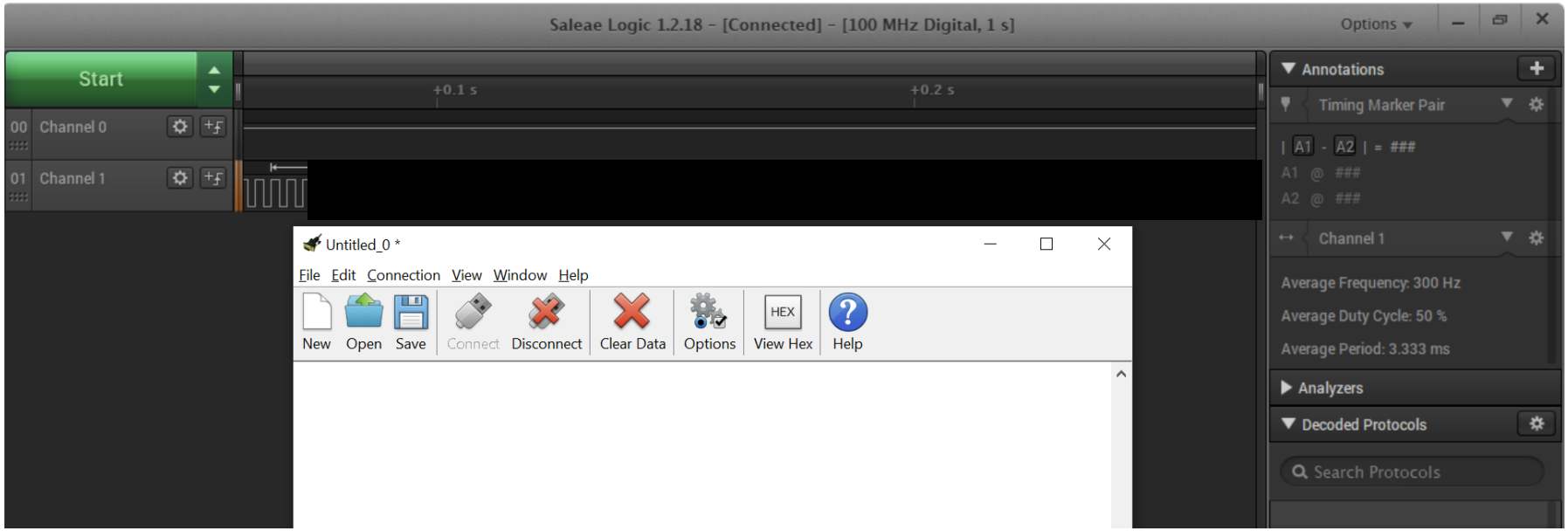
Simplest Serial Data Stream: PWM



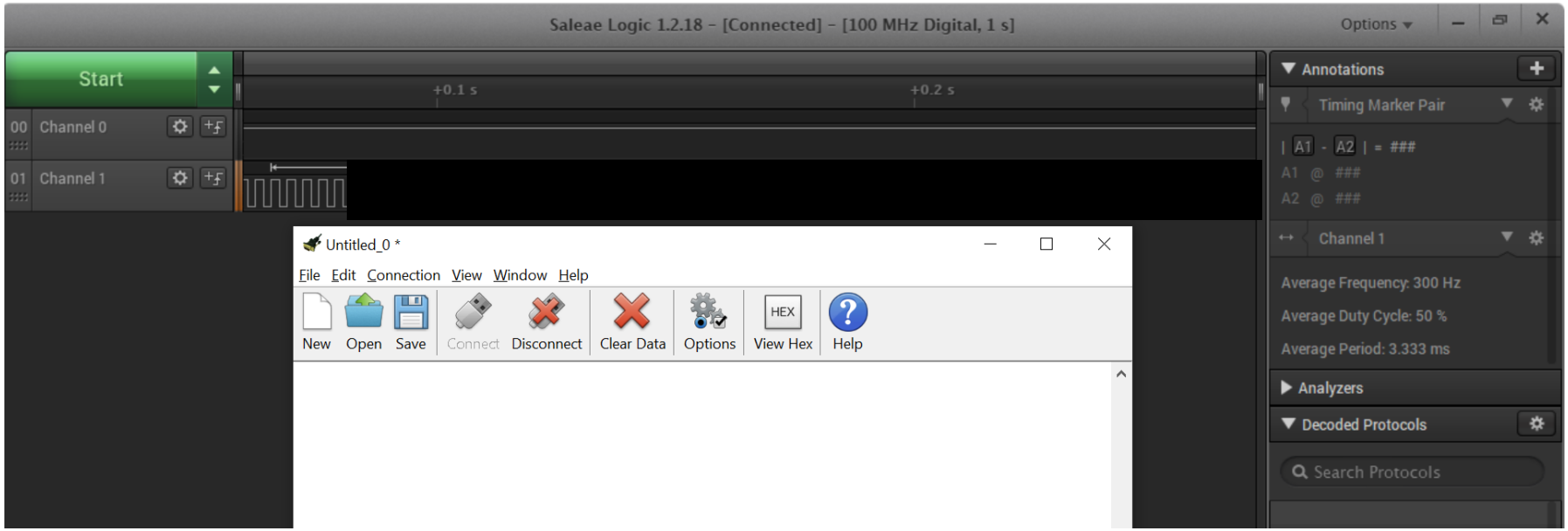
Simplest Serial Data Stream: PWM



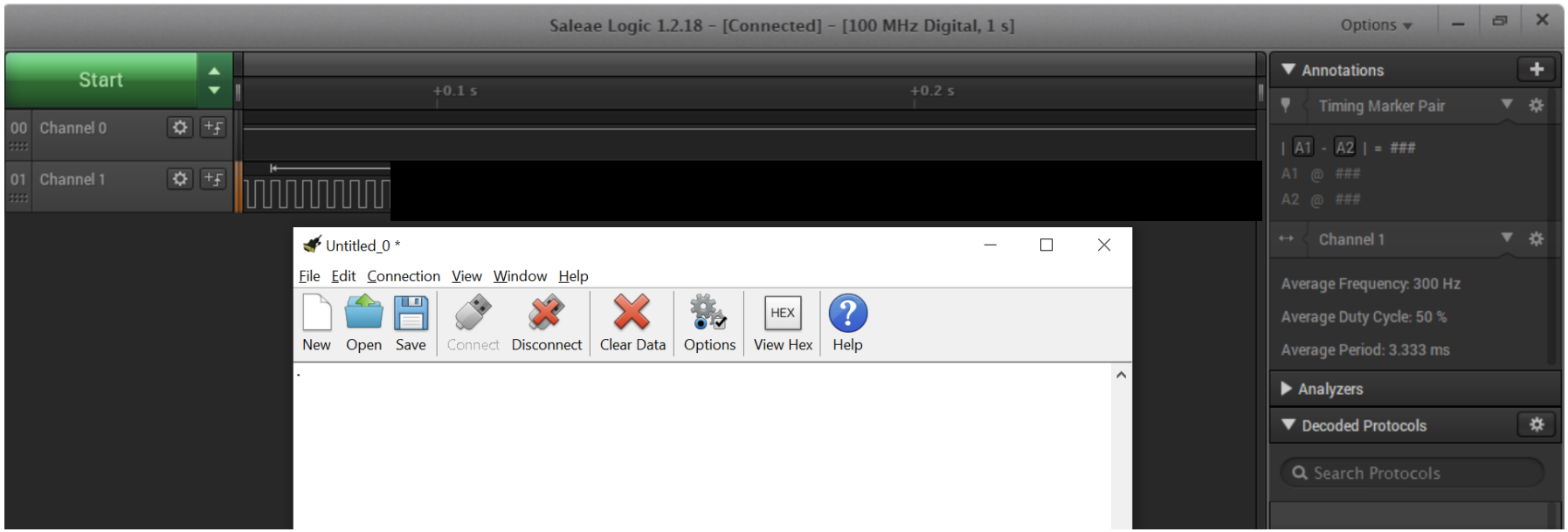
Simplest Serial Data Stream: PWM



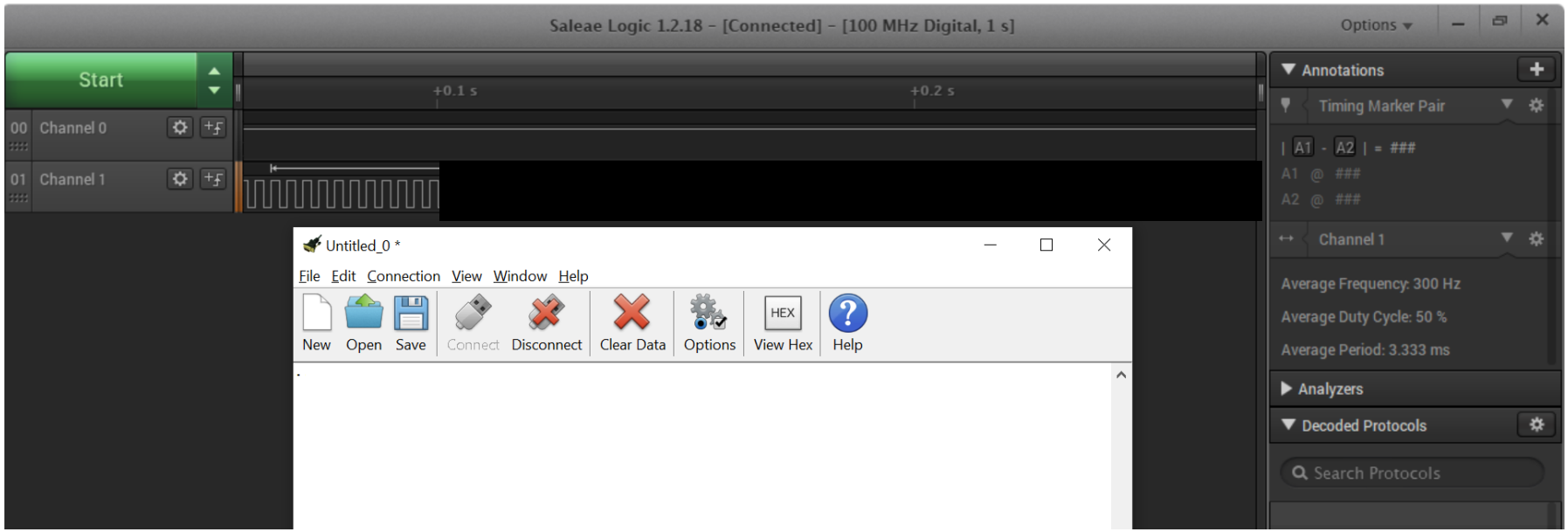
Simplest Serial Data Stream: PWM



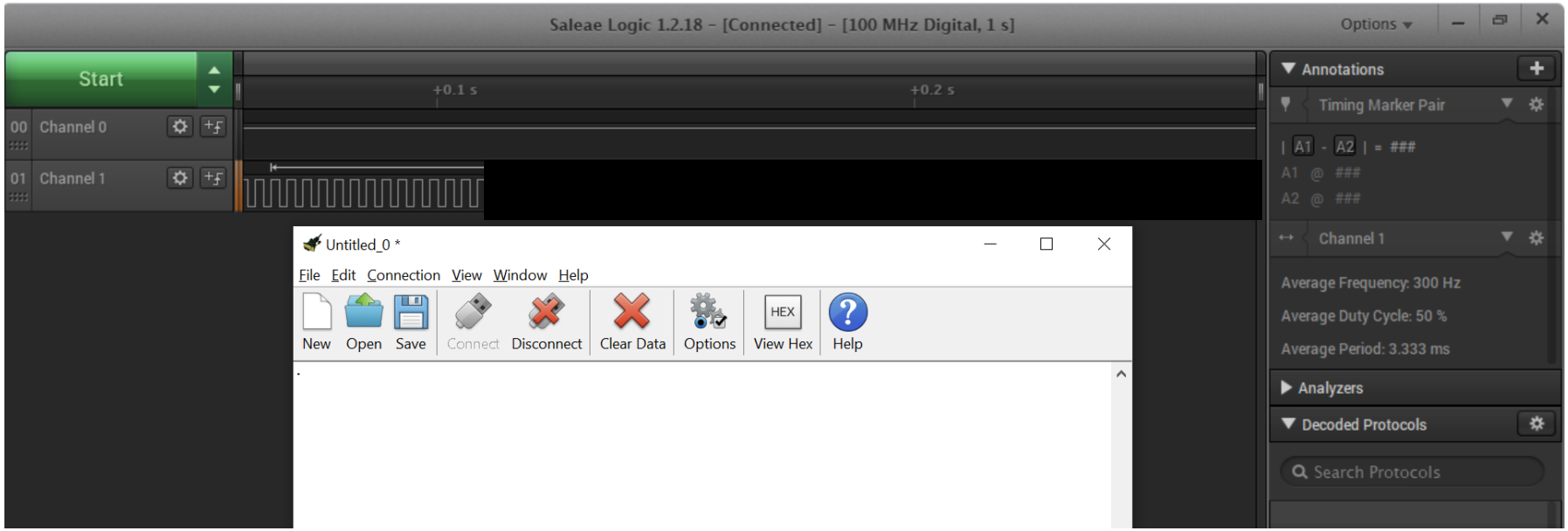
Simplest Serial Data Stream: PWM



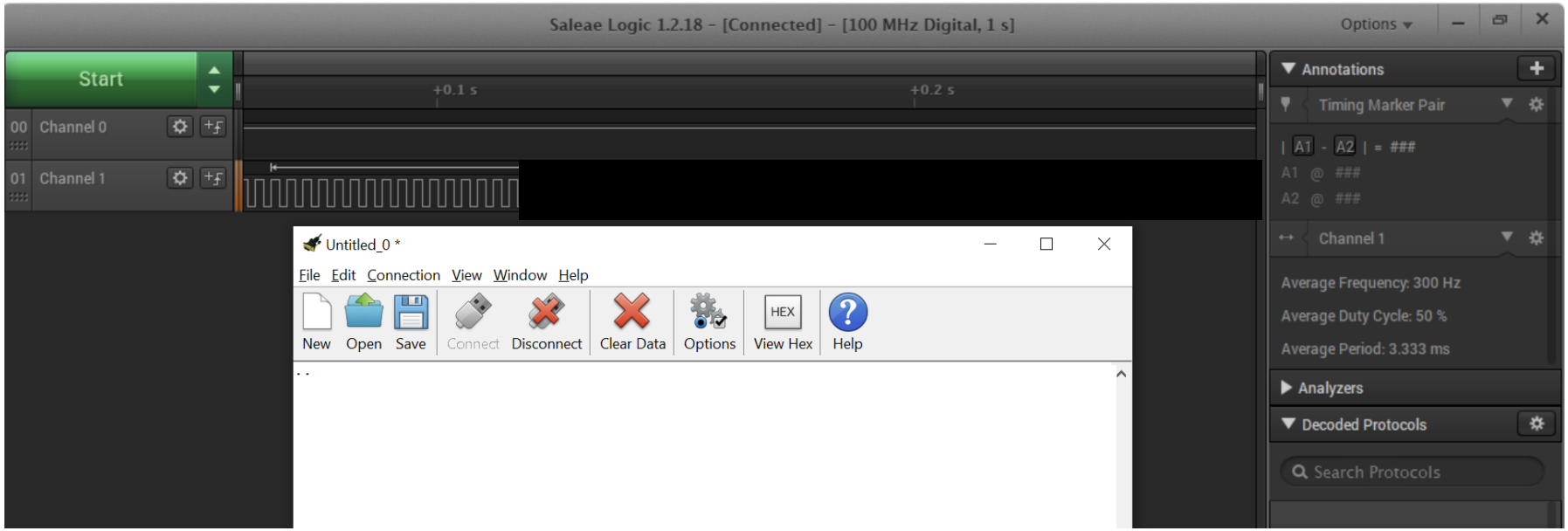
Simplest Serial Data Stream: PWM



Simplest Serial Data Stream: PWM

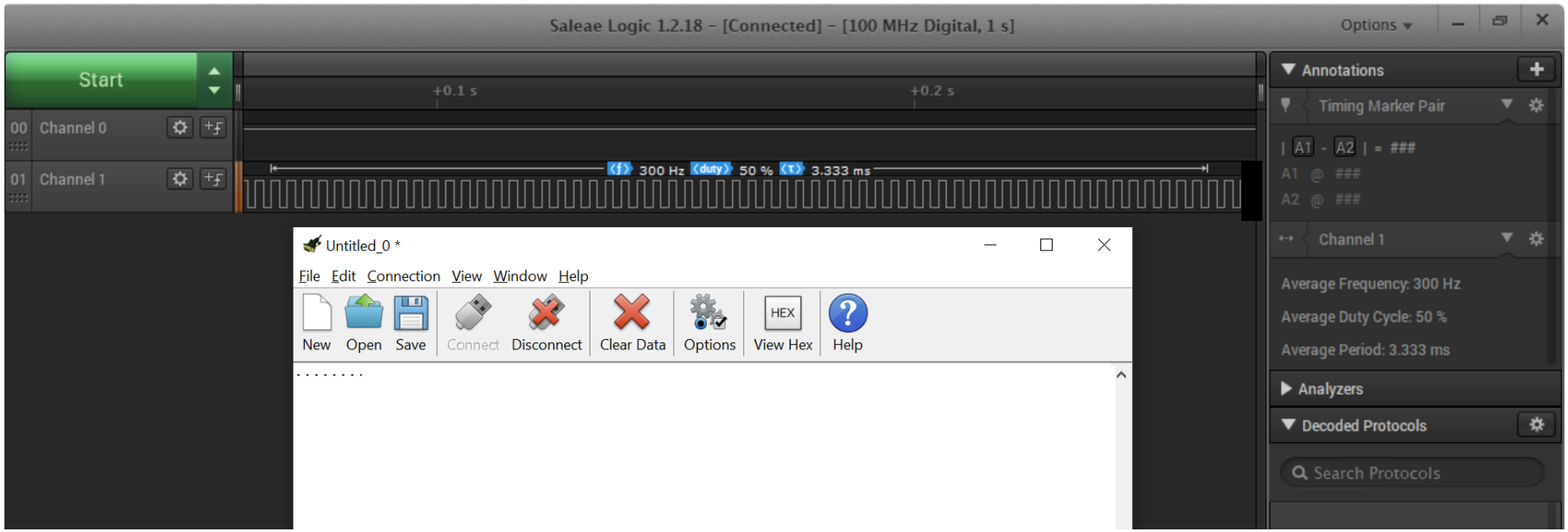


Simplest Serial Data Stream: PWM

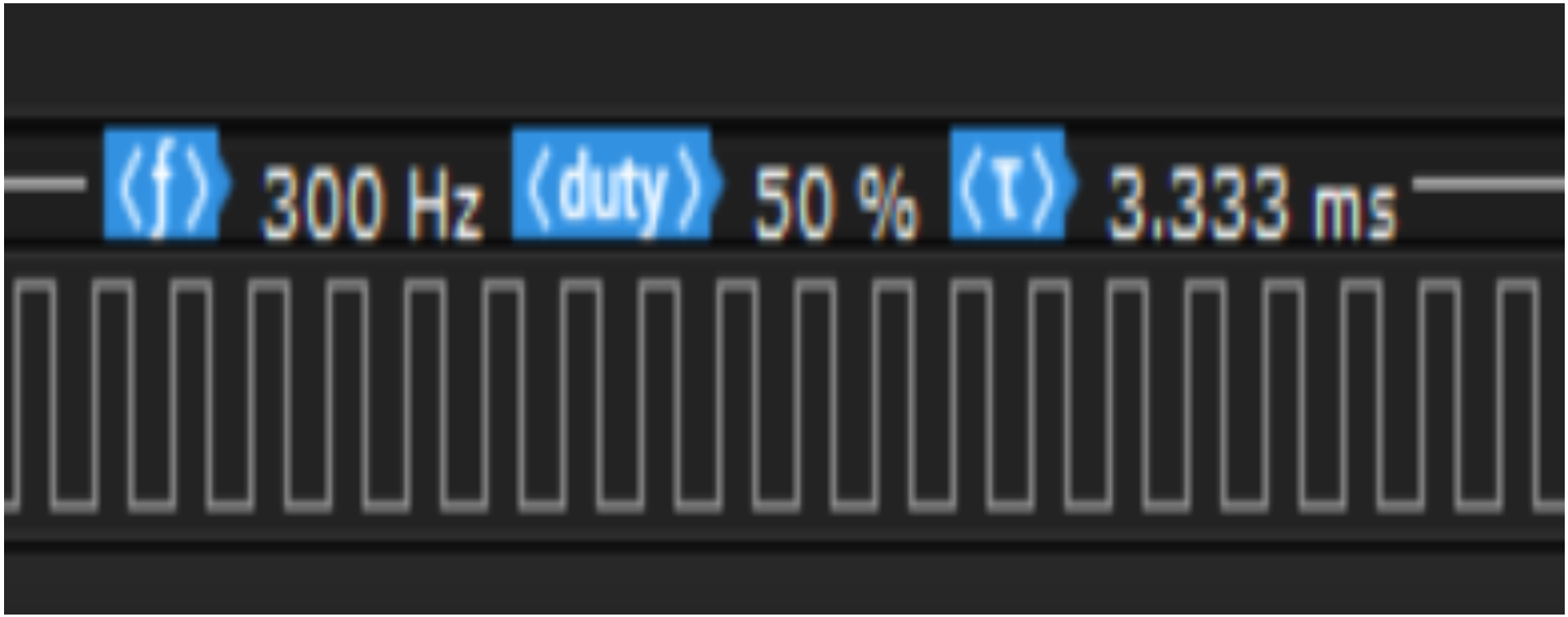


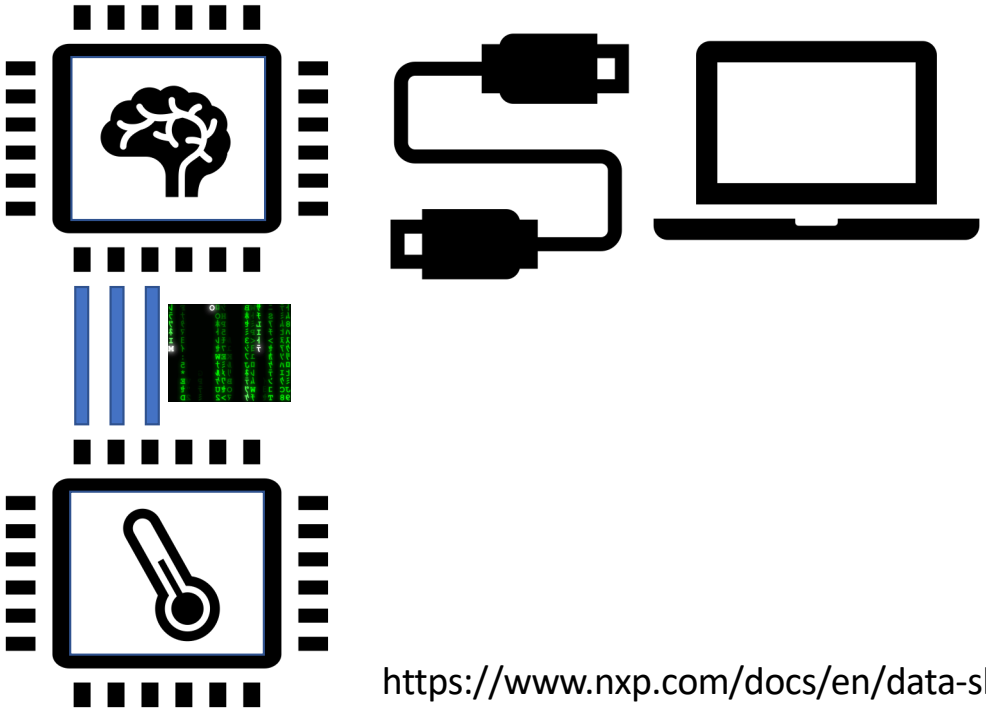
- Keep going …

Simplest Serial Data Stream: PWM



Simplest Serial Data Stream: PWM



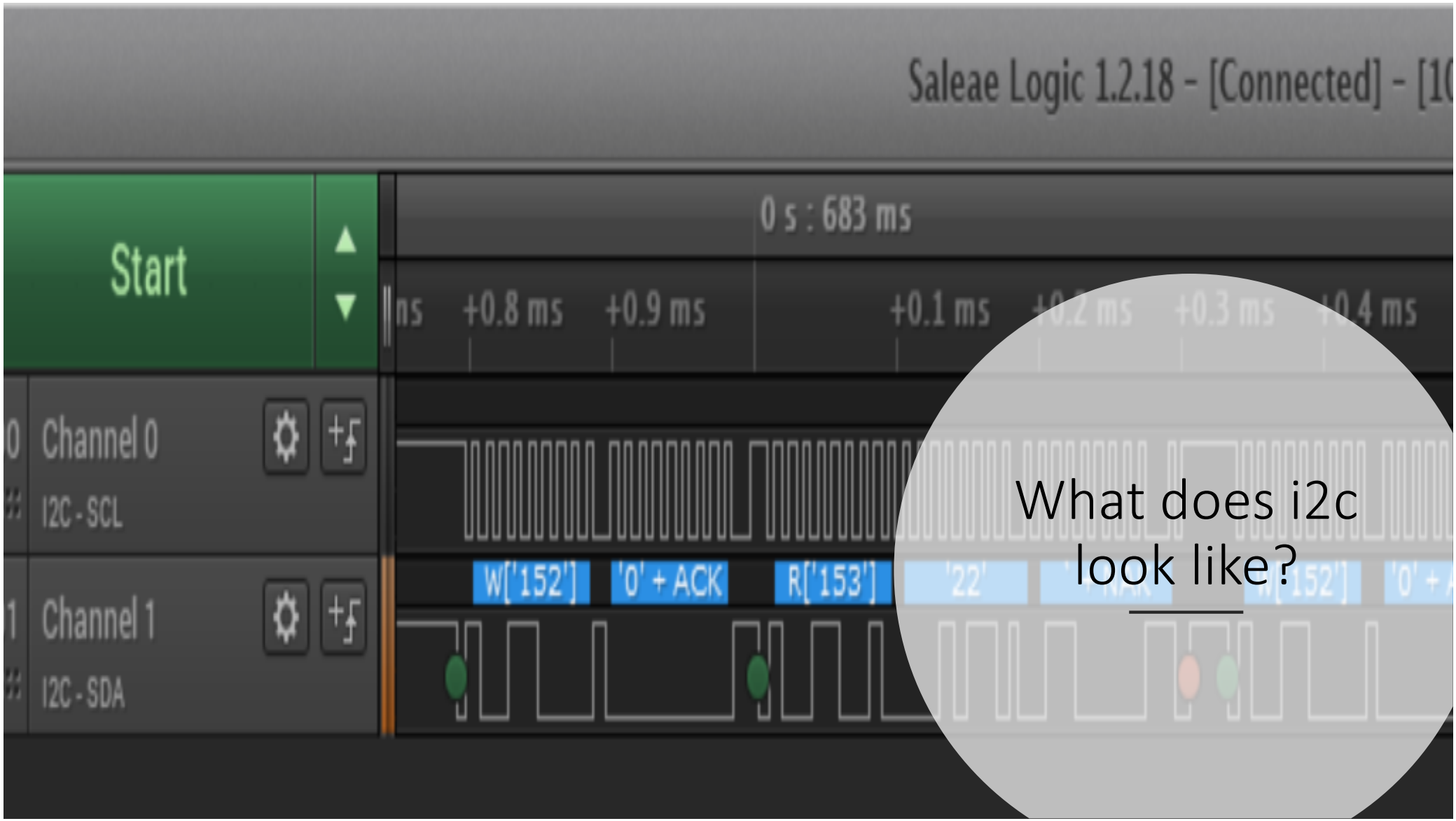


https://www.nxp.com/docs/en/data-sheet/LM75B.pdf

The LM75 Temperature Sensor

…

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

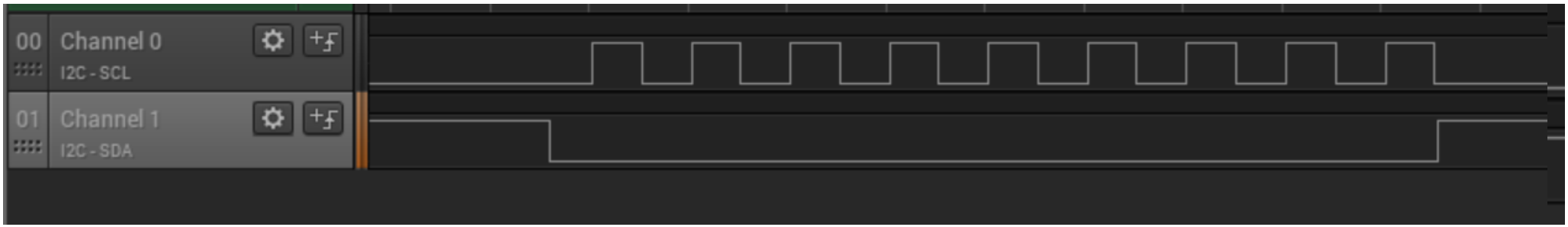

Saleae Logic 1.2.18 - [Connected] - [1
Start
0 s : 683 ms
ns
+0.8 ms
+0.9 ms
+0.1 ms
+0.3 ms
+0.4 ms
Channel 0
I2C - SCL
Channel 1
I2C - SDA
W['152']
'0' + ACK
R['153']
'22'
What does i2c look like?

An I2C message


```
213 void lm75_read(float *data, uint8_t *orgdatabuffe
214 {
215     uint8_t  LM75_Buf[2];
216
217     WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_IDLE
218     I2C0->MSTDAT = (LM75_I2CAddress<<1) | 0;
219     I2C0->MSTCTL = MSTCTL_START;
220
221     WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_TXRD
222     I2C0->MSTDAT = 0x00;
223     I2C0->MSTCTL = MSTCTL_CONTINUE;
```

Message 1

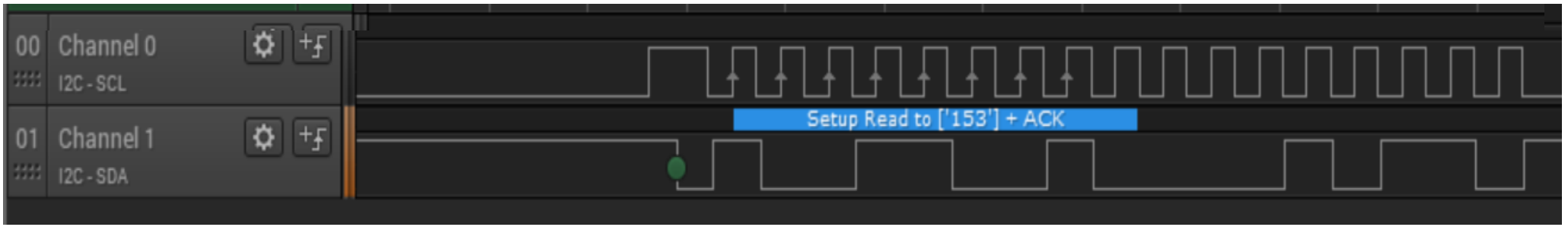
An I2C message (#2)



Message 2

```
197    //    WaitI2CMasterState(I2C0, I2C_STAT_MSTST_TX);   // Wait for the address to be ACK'd
198    WaitI2CMasterState(I2C0, I2C_STAT_MSTST_TXRDY);   // Wait for the address to be ACK'd
199    I2C0->MSTDAT = 0x00;
200    I2C0->MSTCTL = MSTCTL_CONTINUE;
201
202
203    //    WaitI2CMasterState(I2C0, I2C_STAT_MSTST_TX);   // Wait for the address to be ACK'd
204    WaitI2CMasterState(I2C0, I2C_STAT_MSTST_TXRDY);   // Wait for the address to be ACK'd
205    I2C0->MSTDAT = (LM75_I2CAddress<<1) | 1;       // Address with 0 for RWn bit (READ)
206    I2C0->MSTCTL = MSTCTL_START;                   // Start the transaction by setting the MSTSTART bit to 1 in t
```

An I2C message (#3)



Message #3

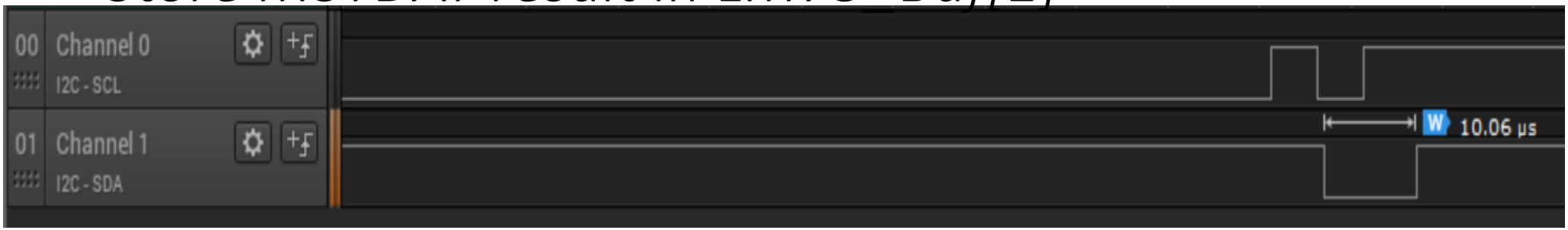
```
225     WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_TXRDY); // Wait f
226     I2C0->MSTDAT = (LM75_I2CAddress<<1) | 1;        // Address
227     I2C0->MSTCTL = MSTCTL_START;                    // Start th
228
229
230     // PRIMARY_STATE_MASK is (0x7<<1)
231         while ((I2C0->STAT & PRIMARY_STATE_MASK) != I2C_STAT_M
232         {
233                 // just spin...
234         }
```

An I2C message (#4): wait for data from Sensor
Store MSTDAT result in LM75_Buf[0]


```
// PRIMARY_STATE_MASK is (0x7<<1)
    while ((I2C0->STAT & PRIMARY_STATE_MASK) != I2C_STAT_MSTST_
    {
            // just spin...
    }

    LM75_Buf[0] = I2C0->MSTDAT;
    I2C0->MSTCTL = MSTCTL_CONTINUE;
    WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_RXRDY);    // Wait
    // PRIMARY_STATE_MASK is (0x7<<1)
```

Message 4

An I2C message (#5): wait for data from Sensor
Store MSTDAT result in LM75_Buf[1]



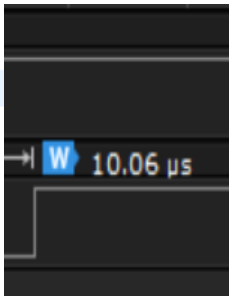

```
237         I2C0->MSTCTL = MSTCTL_CONTINUE;
238         WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_RXRDY);   // Wait for th
239         // PRIMARY_STATE_MASK is (0x7<<1)
240
241         while ((I2C0->STAT & PRIMARY_STATE_MASK) != I2C_STAT_MSTST_RXRDY)
242         {
243                 // just spin...
244         }
245
246         LM75_Buf[1] = I2C0->MSTDAT;
247         I2C0->MSTCTL = MSTCTL_STOP;
248
249
250         // PRIMARY_STATE_MASK is (0x7<<1)
251            while ((I2C0->STAT & PRIMARY_STATE_MASK) != I2C_STAT_MSTST_IDL
```

An I2C message (#5): *Buffers are ready*

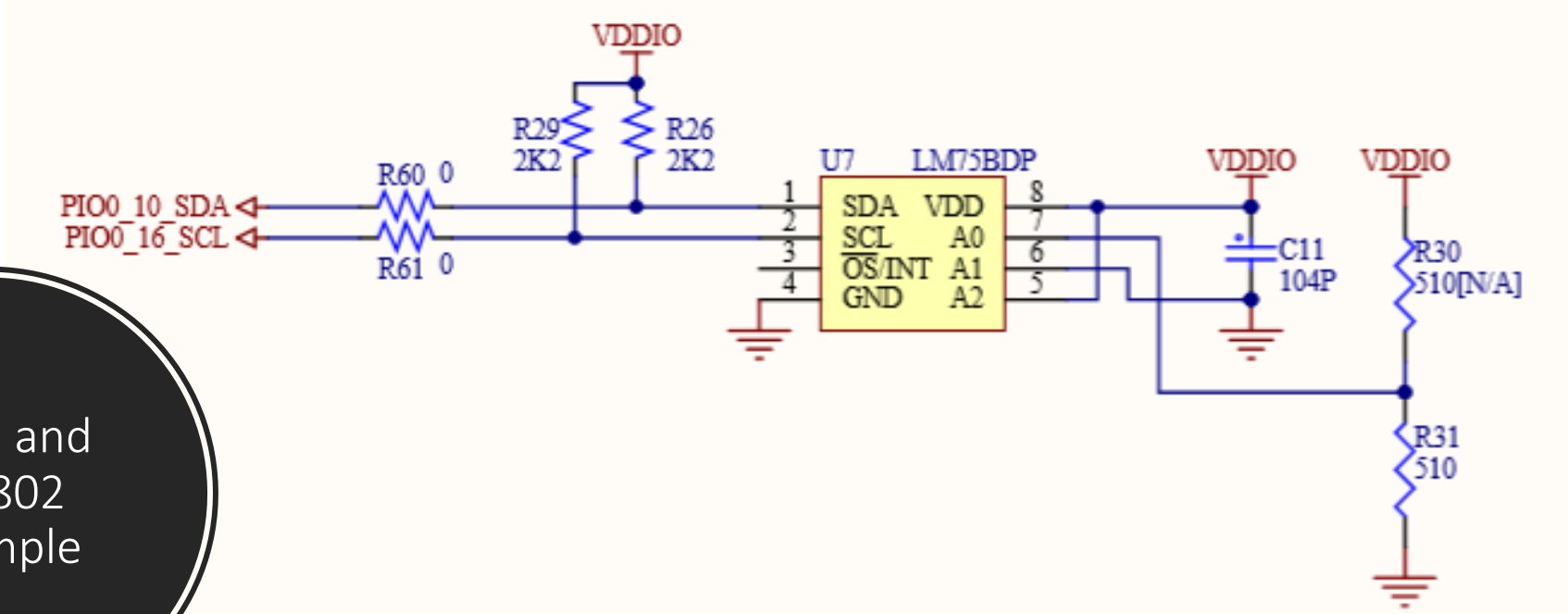

```
222     {
223         // just spin...
224     }
225     LM75_Buf[1] = I2C0->MSTDAT;
226
227
228 //
229
230
231
232
233
234 #if
235 /* L
236 //
```

Expression	Type	Value
LM75_Buf	uint8_t [2]	268437468
LM75_Buf[0]	uint8_t	22
LM75_Buf[1]	uint8_t	0

```
Name : LM75_Buf
    Details:"\026"
    Default:0x100007dc
    Decimal:268437468
    Hex:0x100007dc
    Binary:10000000000000000011111011100
    Octal:02000003734
```



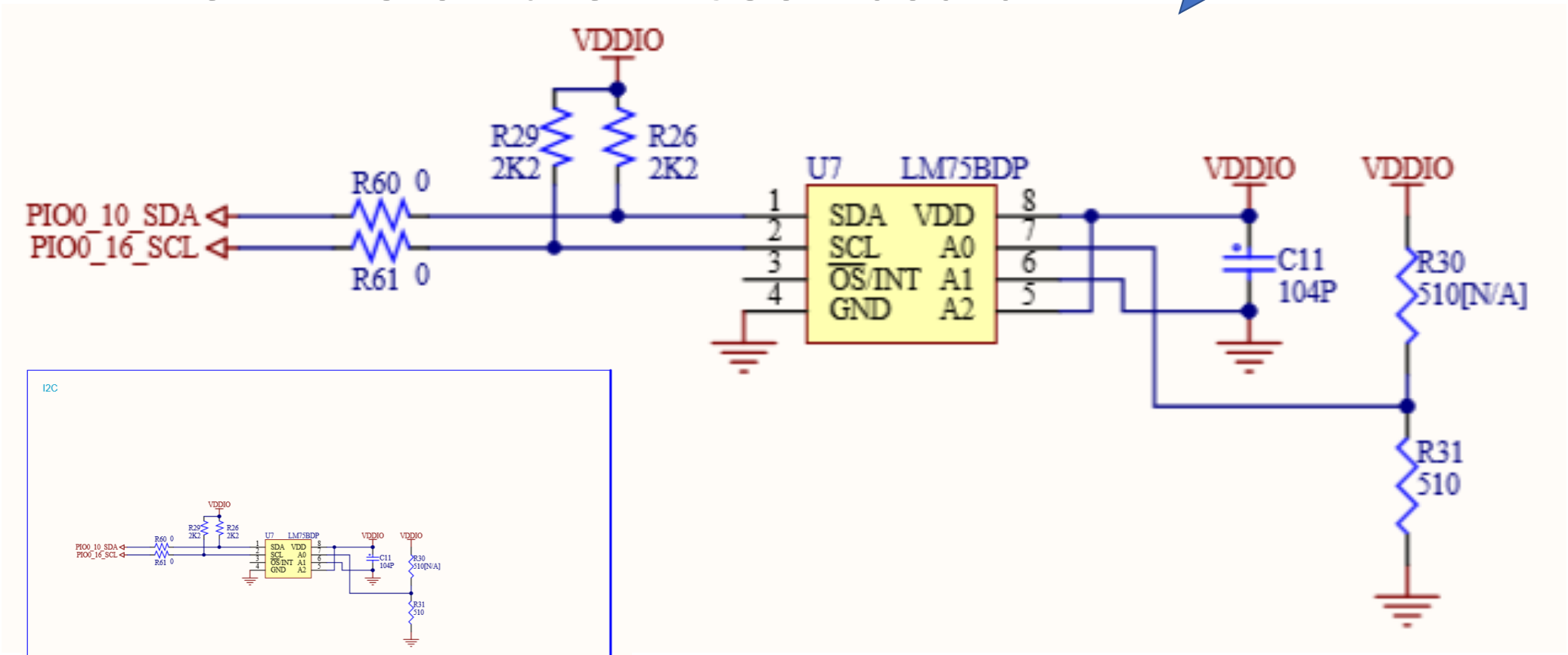
LM75 and LPC802 Example



* Read the LM75 temperature sensor from LPC802 via i2c.
* Display the temperature using a variable frequency signal to an LED.
* Use the Wakeup Timer (WKT) to vary the frequency. (PIO0_8)
* Use the SysTick Timer to sample temperature on a regular basis.
* Use SysTick to blink a heartbeat, too. (PIO0_9)

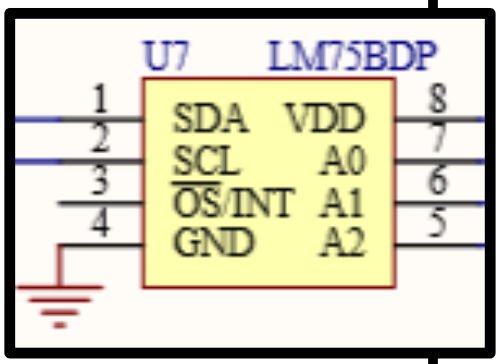
The LM75 on the LPC802 board

Fill in here



LM802 Schematic from NXP

LM75



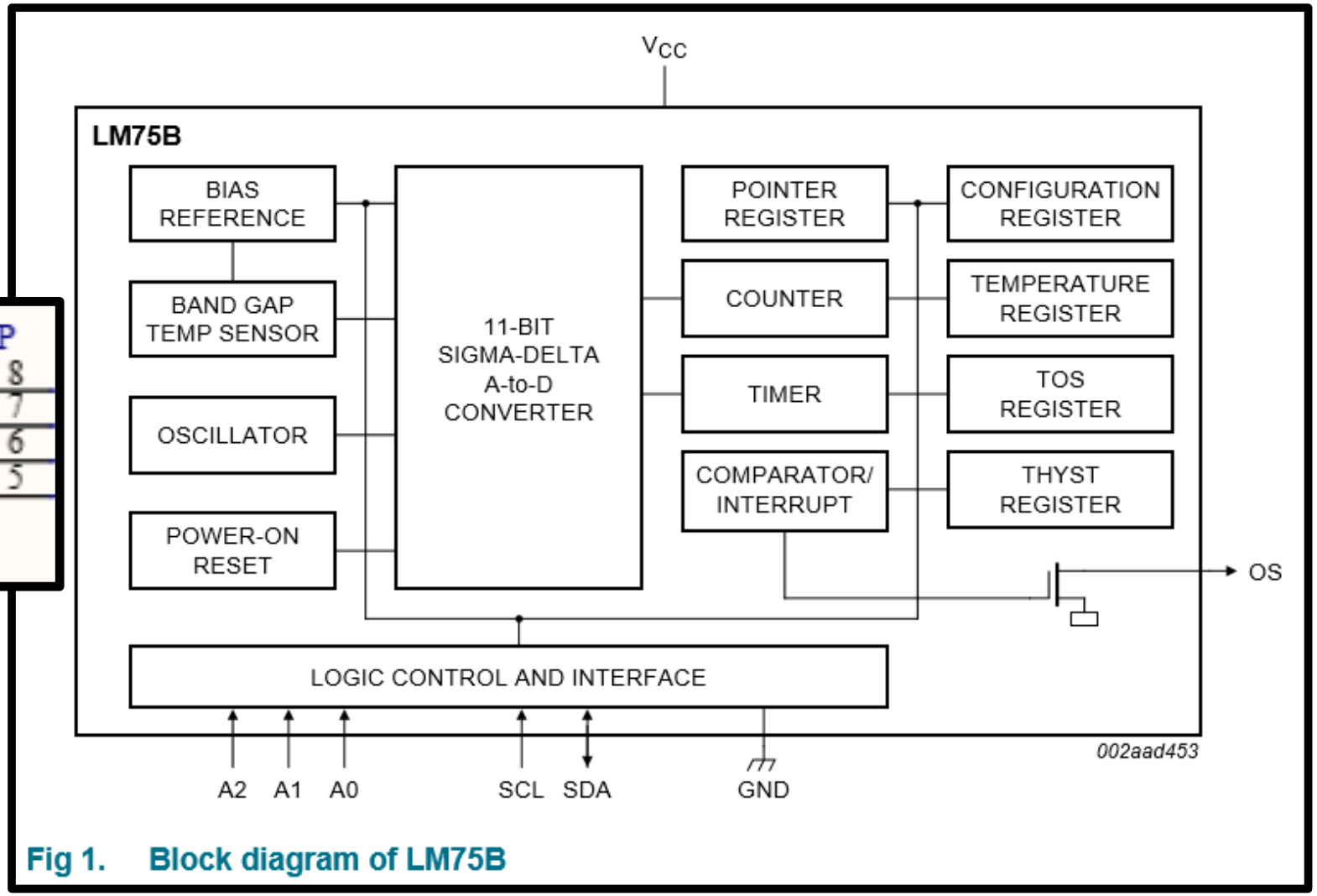


Fig 1. Block diagram of LM75B

https://www.nxp.com/docs/en/data-sheet/LM75B.pdf

LM75

Table 10. Temp register value

11-bit binary (two's complement)	Hexadecimal value	Decimal value	Value
011 1111 1000	3F8	1016	+127.000 °C
011 1111 0111	3F7	1015	+126.875 °C
011 1111 0001	3F1	1009	+126.125 °C
011 1110 1000	3E8	1000	+125.000 °C
000 1100 1000	0C8	200	+25.000 °C
000 0000 0001	001	1	+0.125 °C

https://www.nxp.com/docs/en/data-sheet/LM75B.pdf

LM75: SDA Signal & SCL Clock

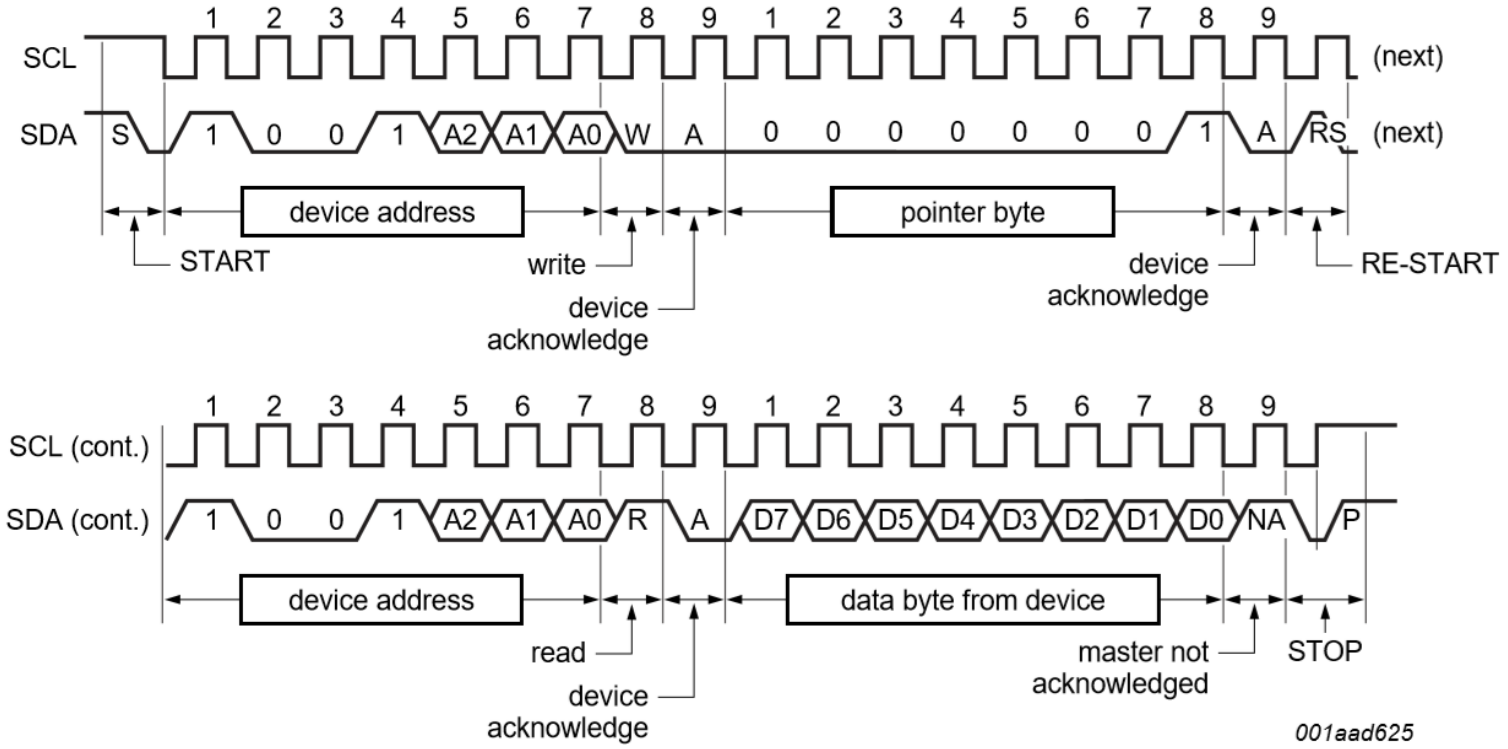
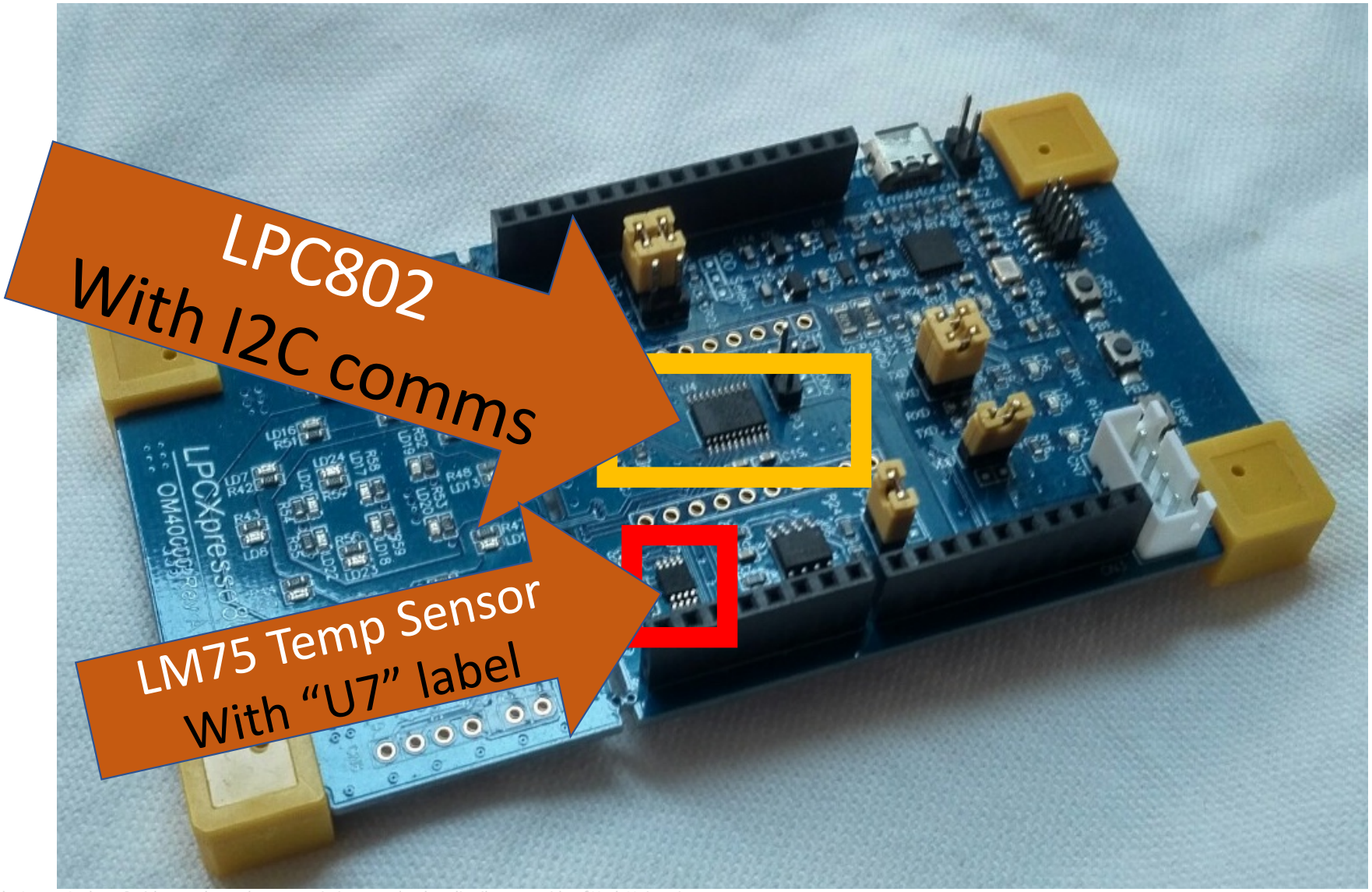
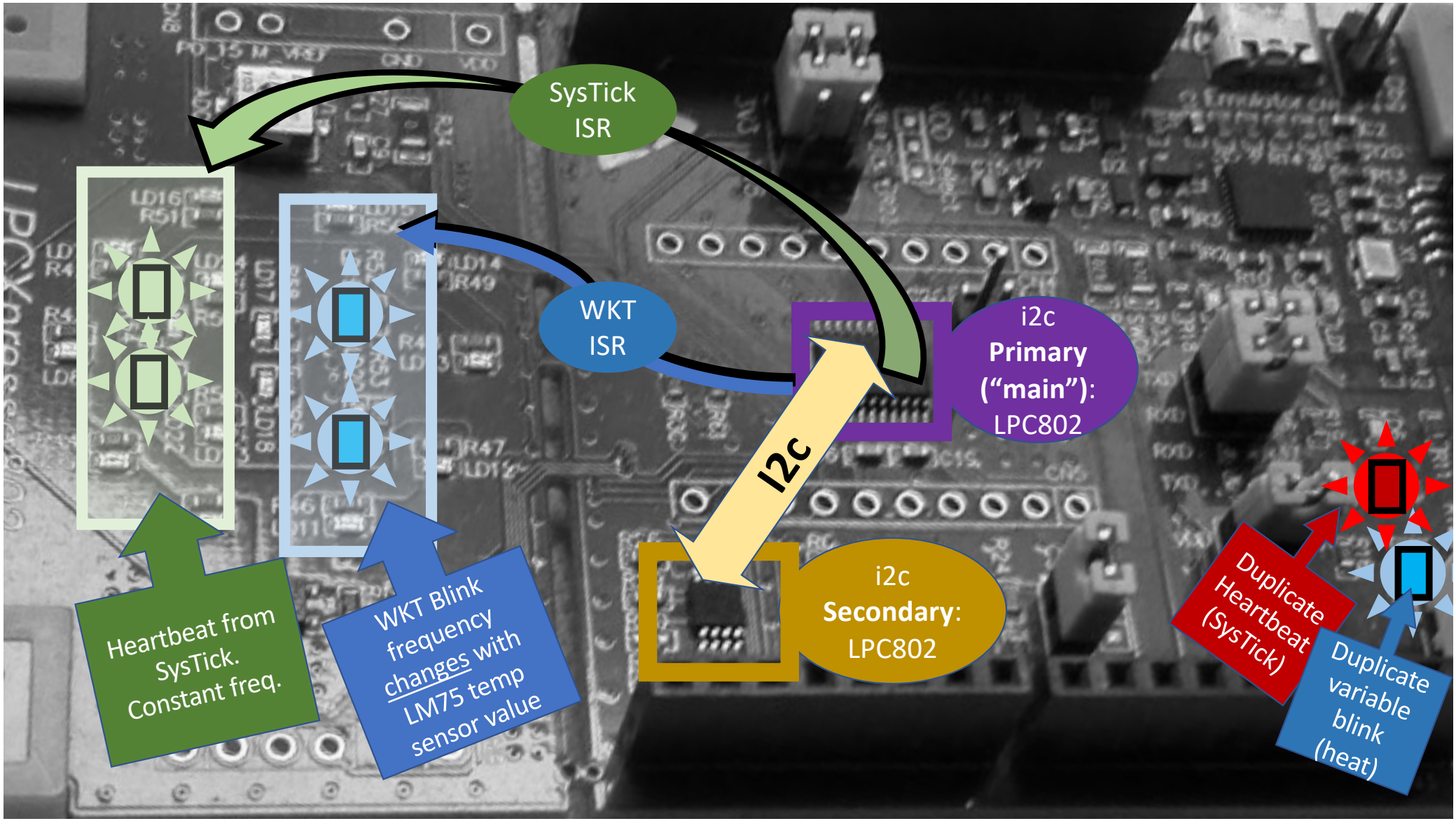


Fig 8. Read configuration register including pointer byte (1-byte data)

https://www.nxp.com/docs/en/data-sheet/LM75B.pdf




SysTick
ISR
WKT
ISR
i2c
Primary
("main"):
LPC802
i2c
i2c
Secondary:
LPC802
Heartbeat from
SysTick.
Constant freq.
WKT Blink
frequency
changes with
LM75 temp
sensor value
Duplicate
Heartbeat
(SysTick)
Duplicate
variable
blink
(heat)

Setup (0 of 4)

```
/**
 * @file    LPC802_Project_3215_i2c_v2.c
 *
 * Use the LM75 temperature sensor via i2c.
 * Display the temperature using a variable frequency
 * signal to an LED.
 * Use the Wakeup Timer (WKT) to vary the frequency. (PIO0_8)
 * Use the SysTick Timer to sample the temperature on a regular basis. (PIO0_9)
 *
 * SDA is PIO0_10; SCL is PIO0_16 to the i2c temp sensor LM75
 *
 * Sources: NXP SDK for LPC802 & NXP Demo Code set for LPC802 OM40000 board.
 *
 * Inaccurate & racist terminology explicitly replaced.
 * See discussion (https://bit.ly/2THLd0b;
https://www.theregister.co.uk/2018/09/11/python_purges_primary_and_slave_in_political_pogrom/)
 *
 * Explictly use Primary to refer to the main driving system (like the '802 processor) and
 * the Secondary is the sensor or thing that is driven (like the LM75 sensor)
 */
```

Setup (1 of 4)

```
#include "clock_config.h"
#include "LPC802.h"
#include <math.h>


// Defines


#define LED_USER1 (8)// use with WKT
#define LED_USER2 (9)// use with SysTick
```

Setup (2 of 4)

```
#define LPC_I2C0BUFFERSize          (35)
#define LPC_I2C0BAUDRate            (100000)// 100kHz

volatile uint8_t  g_I2C0DataBuf[LPC_I2C0BUFFERSize];
volatile uint8_t  g_I2C0DataCnt;

#define WKT_FREQ 1000000          // Use if the WKT is clocked by the LPOSC
#define WKT_RELOAD 100000         // Reload value for the WKT down counter

uint32_t g_WKT_RELOAD = WKT_RELOAD; // counter reload value for WKT

#define LM75_I2CAddress (0x4C)

// Define values for I2C registers that aren't in the header file.
// Table 195 of LPC802 User Manual
#define MSTCTL_CONTINUE (1UL << 0)      // Bit 0 of MSTCTL set ("Main" or "Primary")
#define MSTCTL_START (1UL << 1)         // Bit 1 of MSTCTL set ("Main" or "Primary")
#define MSTCTL_STOP (1UL << 2)          // Bit 2 of MSTTCL set ("Main" or "Primary")
#define CTL_SLVCONTINUE (1UL << 0)      // Bit 0: Secondary level (SLV) Continue
#define CTL_SLVNACK (1UL << 1)          // Bit 1: Secondary Level (SLV) Acknowledge
```

Setup (3 of 4)

```
#define PRIMARY_STATE_MASK (0x7<<1) // bits 3:1 of STAT register for Main / Primary
#define I2C_STAT_MSTST_IDLE ((0b000)<<1) // Main Idle: LPC802 user manual table 187
#define I2C_STAT_MSTST_RXRDY ((0b001)<<1) // Main Receive Ready          " "
#define I2C_STAT_MSTST_TXRDY ((0b010)<<1) // Main Transmit Ready         " "
#define I2C_STAT_MSTST_NACK_ADD ((0b011)<<1)// Main Ack Add              " "
#define I2C_STAT_MSTST_NACK_DATA ((0b100)<<1)// Main Ack signal data     ” ”


// Prototypes of Functions
void   WaitI2CPrimaryState(I2C_Type * ptr_LPC_I2C, uint32_t state);
void   I2C_PrimarySetBaudRate(uint32_t baudRate_Bps, uint32_t srcClock_Hz);
void   WKT_Config(void);
void  lm75_i2c_init(void);
void  lm75_read(float *data, uint8_t *orgdatabuffer);
```

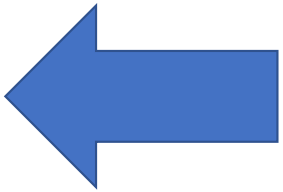
Setup (4 of 4)

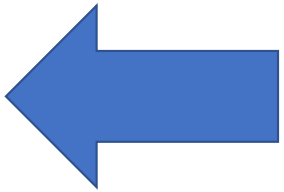
```
float    g_LM75SensorValue              = 0;
uint32_t g_LM75SensorValueInt           = 0;
uint8_t  g_LM75SensorValueChar[4]       = {0};
uint8_t  g_LM75SensorData[2]            = {0};



volatile uint32_t g_SystemTicks         = 0;
```

SysTick ISR (Part 1 of 2)
Read the temperature sensor at intervals

Fill in here

```
/* System Tick handler: 2 jobs
 * 1. Toggle the heartbeat.
 * 2. Load the Wakeup Timer's value proportional to LM75 sensor value.
 */
void SysTick_Handler(void) {

  // System Tick ++


  //interval display the value
  if( g_SystemTicks >= 5 )
  {
      // Read Temperature value from LM75
                    (&g_LM75SensorValue, g_LM75SensorData);

      // Toggle the LED with the NOT register.
      GPIO->NOT[0] = (1UL<<LED_USER2);// LED is on PIO0_9
```





SysTick ISR (Part 2 of 2)
Modify WKT freq. based on temperature

```
// Set the WKT counter reload value based on the temperature reported by LM75
// Sensor returns 20 – 30 around room temperature.
if (g_LM75SensorValue <= 20)
{
	g_WKT_RELOAD = (uint32_t)(WKT_RELOAD*20);
}
else if (g_LM75SensorValue <= 22){
	g_WKT_RELOAD = (uint32_t)(WKT_RELOAD*10);
}
else if (g_LM75SensorValue <= 24){
	g_WKT_RELOAD = (uint32_t)(WKT_RELOAD*5);
}
else if (g_LM75SensorValue <= 26){
	g_WKT_RELOAD = (uint32_t)(WKT_RELOAD*1);
}
else if (g_LM75SensorValue <= 28){
	g_WKT_RELOAD = (uint32_t)(WKT_RELOAD*0.5);
}
else{ // it's above 29
	g_WKT_RELOAD = (uint32_t)(WKT_RELOAD*0.1);
}
	g_SystemTicks = 0;// reset the number of ticks
}
}
```

Init i2c (part 1 of 5)
Assign pins via switch matrix

```
// ------------------------------------------------------------
//
// Init the LM75 sensor for i2c.
// originally from SDK for LPC802 and NXP demo code
void lm75_i2c_init(void){
    // ------------------------------------------------------------
    // Step 1: Connect I2C module to outer pins via Switch Matrix
    // PIO0_16 is connected to the SCL line
    // PIO0_10 is connected to the SDA line
    // PINASSIGN5 bits 15:8 are for SCL.  Therefore fill with value 16
    // PINASSIGN5 bits 7:0 are for SDA. Therefore fill with value 10
    //
    // enable switch matrix.
    SYSCON->SYSAHBCLKCTRL0 |= (SYSCON_SYSAHBCLKCTRL0_SWM_MASK);

    // Set switch matrix
    // PINASSIGN5: clear bits 15:0 to permit the two i2c lines to be assigned.
    SWM0->PINASSIGN.PINASSIGN5 &= ~(SWM_PINASSIGN5_I2C0_SCL_IO_MASK |
                                    SWM_PINASSIGN5_I2C0_SDA_IO_MASK);          // clear 15:0
    SWM0->PINASSIGN.PINASSIGN5 |= ((16<<SWM_PINASSIGN5_I2C0_SCL_IO_SHIFT)|    // Put 16 in bits 15:8
                                    (10<<SWM_PINASSIGN5_I2C0_SDA_IO_SHIFT));   // put 10 in bits 7:0
    // PINASSIGN5 should be 0xffff100a after this.
    // disable the switch matrix
    SYSCON->SYSAHBCLKCTRL0 &= ~(SYSCON_SYSAHBCLKCTRL0_SWM_MASK);
    // ---------------- End of Switch Matrix code ---------------------------------
```

Init i2c (part 2 of 5)
Turn on i2c module

```
// ------------------------------------------------------------
// Step 2: Turn on the I2C module via the SYSCON clock enable pin
// ------------------------------------------------------------


// debug: just i2c
SYSCON->SYSAHBCLKCTRL0 |= (SYSCON_SYSAHBCLKCTRL0_I2C0_MASK  );// I2C is on


// debug: in the demo code they don't seem to reset the i2c clock.
// debug: just i2c
// Put 0 in the GPIO, GPIO Interrupt and I2C reset bit to reset it.
// Then put a 1 in the GPIO, GPIO Interrupt and I2C reset bit to allow them to operate.
// manual: Section 6.6.10
SYSCON->PRESETCTRL0 &= ~(SYSCON_PRESETCTRL0_I2C0_RST_N_MASK );// reset  I2C(bit = 0)
SYSCON->PRESETCTRL0 |= (SYSCON_PRESETCTRL0_I2C0_RST_N_MASK);// remove i2C reset (bit = 1)
```

Init i2c (part 3 of 5)
choose source of i2c timing

```
// ----------------------------------------------------------
// Step 3: Choose the source of the I2C timing clock
// see: Fig 7 of User Manual
// ----------------------------------------------------------


// Provide main_clk as function clock to I2C0
// bits 2:0 are 0x1.
// FRO is 0b000; main clock is 0b001, frg0clk is 0b010, fro_div is 0b100.
// Set i2c to FRO (12 MHz)
// (only bits 2:0 are used; other 29 bits are ignored)
SYSCON->I2C0CLKSEL = 0b000;// put 000 in bits 2:1.
// confirmed in reg view that this is FRO.
```

Init i2c (part 4 of 5)

Set the LPC802 i2c as “primary” (main) on the bus

```
// ---------------------------------------------------------
// Step 4: enable primary (but not slave) functionality in the i2c module.
//
// ---------------------------------------------------------
// Configure the I2C0 CFG register:
// Primary enable = true
// Secondary enable = true
// Monitor enable = false
// Time-out enable = false
// Monitor function clock stretching = false
//
// Only config I2C0 as a primary
I2C0->CFG = (I2C_CFG_MSTEN_MASK); //  only as primary


// Comment out for now: therefore, no interruptions.
// Enable the I2C0 "secondary" pending interrupt
// I2C0->INTENSET = I2C_INTENSET_SLVPENDINGEN_MASK; // STAT_SLVPEND;
```

Init i2c (part 5 of 5)
Set the transmission frequency of i2c ("baud")

```
    // ------------------------------------------------------------
    // Step 5: set the i2c clock rate. (20Hz to 400kHz is typical)
    // I2C_PrimarySetBaudRate() function is helpful.
    // ------------------------------------------------------------
    // Set i2C bps to 100kHz assuming an input clock of 12MHz
    I2C_PrimarySetBaudRate(100000, 12000000);
    // after this, CLKDIV is 0x9 and MSTTIME is 0x44.
    // MSTTIME = 0x44 means: MSTSCLLOW [2:0] is CLOCKS_6
    //  MSTCLHIGH [6:4] is CLOCKS_6


}
```

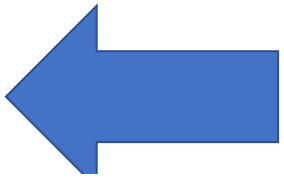
Define reading function for i2c (1 of 3)

```
// ------------------------------------------------------------
//
// Read the LM75 sensor over i2c.
// originally from SDK for LPC802 and NXP demo code
void lm75_read(float *data, uint8_t *orgdatabuffer)
{
    uint8_t  LM75_Buf[2];


    WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_IDLE); // Wait for the Primary's state to be idle
    I2C0->MSTDAT = (LM75_I2CAddress<<1) | 0;       // Address with 0 for RWn bit (WRITE)
    I2C0->MSTCTL = MSTCTL_START;                   // Start the transaction by setting the
                                                    // MSTSTART bit to 1 in the Primary control register

    WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_TXRDY);   // Wait for the address to be ACK'd
    I2C0->MSTDAT = 0x00;
    I2C0->MSTCTL = MSTCTL_CONTINUE;

    WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_TXRDY); // Wait for the address to be ACK'd
    I2C0->MSTDAT = (LM75_I2CAddress<<1) | 1;       // Address with 1 for RWn bit (WRITE)
    I2C0->MSTCTL = MSTCTL_START;                   // Start the transaction by setting the
                                                    // MSTSTART bit to 1 in the Primary control register.
```

Define reading function for i2c (2 of 3)

Fill in here

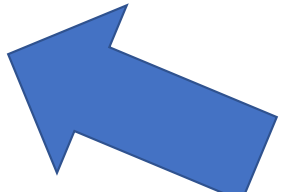
```
// PRIMARY_STATE_MASK is (0x7<<1)
while ((I2C0->STAT & PRIMARY_STATE_MASK) != I2C_STAT_MSTST_RXRDY)
{
       // just spin... (wait)
}

LM75_Buf[0] = I2C0->MSTDAT;
I2C0->MSTCTL = MSTCTL_CONTINUE;
// Wait for the address to be ACK'd
WaitI2CPrimaryState(I2C0, I2C_STAT_MSTST_RXRDY);
// PRIMARY_STATE_MASK is (0x7<<1)

while ((I2C0->STAT & PRIMARY_STATE_MASK) != I2C_STAT_MSTST_RXRDY)
{
     // just spin... (wait)
}

LM75_Buf[1] = I2C0->
I2C0->MSTCTL = MSTCTL_STOP;
```



Define reading function for i2c (3 of 3)
Convert the data into usable form.

Fill in here

```
    // PRIMARY_STATE_MASK is (0x7<<1)
    while ((I2C0->STAT & PRIMARY_STATE_MASK) != I2C_STAT_MSTST_IDLE)
    {
    // just spin...
    }

    // Convert the buffer into a single variable
    // accessible outside this function via the *data pointer.

    if( (LM75_Buf[0]&0x80) == 0x00){
        *data = ((float)( ((LM75_Buf[0]<<8) + LM75_Buf[1])>>5) * 0.125);
    }
    else{
        *data = 0x800 - ((                [0]<<8) + (                [1]>>5) );
        *data = -(((float)(*data)) * 0.125);
    }
}
```



Main function (part 1 of 5): Disable IRQs

```
// Main routine
// SDA is PIO0_10; SCL is PIO0_16 to the i2c temp sensor LM75
int main(void)
{


    // ------------------------------------------------------------
    // Step 0: disable interrupts
    // ------------------------------------------------------------


    // disable interrupts
    __disable_irq();      // turn off globally
    NVIC_DisableIRQ(SysTick_IRQn);      // turn off the SysTick interrupt.
    NVIC_DisableIRQ(I2C0_IRQn);         // turn off IRQ for I2C
```

Main function (part 2 of 5): FRO (main clock)

```
// ----------------------------------------------------------
// Step 1: Choose & set main clock


// ----------------------- 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); // FRO confirmed.


// 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 5): GPIO

```
// ---------------------- Turn on all modules that we use (clock & reset) --------------------------


SYSCON->SYSAHBCLKCTRL0 |= (SYSCON_SYSAHBCLKCTRL0_GPIO0_MASK);// GPIO is on


// Put 0 in the GPIO, GPIO Interrupt and I2C reset bit to reset it.
// Then put a 1 in the GPIO, GPIO Interrupt and I2C reset bit to allow them to operate.
// manual: Section 6.6.10
SYSCON->PRESETCTRL0 &= ~(SYSCON_PRESETCTRL0_GPIO0_RST_N_MASK);// reset  GPIO (bit = 0)
SYSCON->PRESETCTRL0 |= (SYSCON_PRESETCTRL0_GPIO0_RST_N_MASK);// remove GPIO reset(bit = 1)


// ---------------------- Begin GPIO setup -----------------------------------
// Config the LED (GPIO 8) and LED (GPIO 9) for output
// Remember: only bits set to 1 have an effect on DIRCLR and DIRSET registers.
//    bits cleared to 0 are ignored.
// Therefore, use DIRCLR to select input and DIRSET to select output
GPIO->CLR[0] = (1UL<<LED_USER1);// LED is on PB8(     turn off   )
GPIO->DIRSET[0] = (1UL<<LED_USER1);// output LED on PIO0_8


GPIO->CLR[0] = (1UL<<LED_USER2);// LED is on PB9(     turn off   )
GPIO->DIRSET[0] = (1UL<<LED_USER2);// output on PB9 (LED_USER2)


// ---------------------- end of GPIO setup ----------------------------------
```

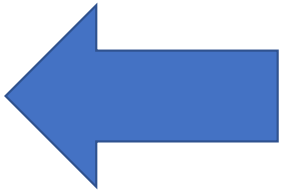
Main function (part 4 of 5): SysTick Heartbeat

```
// ------------------------ Begin SysTick setup --------------------------
// Configure SysTick to fire once every 0.2 seconds.
// Argument is the number of clock ticks between IRQs.
//
// Clock rate: 12 MHz via FRO. (8.3*10^(-8) sec period)
// 12,000,000 ticks/sec * 0.2 sec = 2400000 ticks
// SysTick is a 24 bit timer (max almost 17 million)


SysTick_Config(2400000);  // 2400000 ticks = 0.2 sec @ 12 MHz


// ------------------------ end SysTick setup ----------------------------
```

Main function (part 5 of 5):
Init LM75, Init WKT, enable IRQs and Loop.

Fill in here

```
    // Initialize LM75 temperature sensor, I2C
    lm75_i2c_init();

    // init the WKT



    // enable interrupts
    __enable_irq();      // turn on globally
    NVIC_EnableIRQ(SysTick_IRQn); // turn on the SysTick interrupt.


    while(1)
    {
        // do nothing, just keep going.
        __asm("NOP");
    };
}

// end of main
```



i2c Wait on Primary (part 1 of 2)

```
/*****************************************************************************
** Function name:WaitI2CPrimaryState
**
** Description:    Waits for I2C primary pending, then compares the primary
**                      state to the state parameter. If compare fails, enter
**                      a while(1) loop. If compare passes, return.
**
** parameters:
**                      ptr_LPC_I2C: A pointer to an I2C instance
**                      state: One of the 3-bit Primary function state codes of the
I2C
** Returned value:None
**
** originally from SDK / Demo Code for LPC802 / OM40000;
*****************************************************************************/
void WaitI2CPrimaryState(I2C_Type * ptr_LPC_I2C, uint32_t state) {
```

i2c Wait on Primary (part 2 of 2)

```
    // Check the Primary Pending bit (bit 0 of the i2c stat register)

    // Wait for MSTPENDING bit set in STAT register
    while(!(ptr_LPC_I2C->STAT & I2C_STAT_MSTPENDING_MASK)); // Wait

    // Check to see that the state is correct.
    // if it is not, then turn on PIO0_9 to indicate a problem
    // Primary's state is in bits 3:1. PRIMARY_STATE_MASK is (0x7<<1)
    if((ptr_LPC_I2C->STAT & PRIMARY_STATE_MASK) != state)  // If primary state mismatch
    ...
    {
        GPIO->DIRCLR[0] = (1UL<<LED_USER2);     // turn on LED on PIO0_9 (LED_USER2)
        while(1);                               // die here and debug the problem
    }

    return;                         // If no mismatch, return

}
```

Set I2C baud rate (Part 1 of 3)
Setup

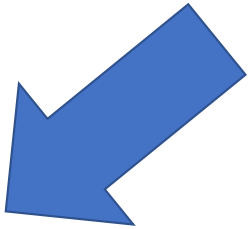
```
// ------------------------------------------------------------------------
// I2C_PrimarySetBaudRate()
//
// originally from SDK for LPC802; fsl_i2c.c
// input: baudRate_Bps & srcClock_Hz
// output: no explicit output.  Two I2C config registers set.
//
// originally from SDK / Demo Code for LPC802 / OM40000;
// ------------------------------------------------------------------------
void I2C_PrimarySetBaudRate(uint32_t baudRate_Bps, uint32_t srcClock_Hz)
{
    uint32_t scl, divider;
    uint32_t best_scl, best_div;
    uint32_t err, best_err;


    best_err = 0;
```

Set I2C baud rate (Part 2 of 3)
check the baud rate for low error

```
 for (scl = 9; scl >= 2; scl--){
        /* calculated ideal divider value for given scl */
        divider = srcClock_Hz / (baudRate_Bps * scl * 2u);


        /* adjust it if it is out of range */
        divider = (divider > 0x10000u) ? 0x10000 : divider;


        /* calculate error */
        err = srcClock_Hz - (baudRate_Bps * scl * 2u * divider);
        if ((err < best_err) || (best_err == 0)){
            best_div = divider;
            best_scl = scl;
            best_err = err;
        }
        if ((err == 0) || (divider >= 0x10000u)){
            /* either exact value was found
               or divider is at its max (it would even greater in the next iteration for sure) */
            break;
        }
}
```

Set I2C baud rate (Part 3 of 3)
set the baud rate

Fill in here

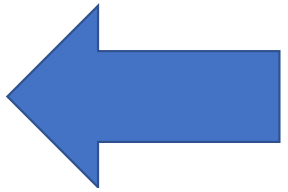


```
// Assign Clock Divider value, using                    included in LPC802.h
I2C0->CLKDIV = I2C_CLKDIV_DIVVAL(best_div - 1);

// Assign Primary timing configuration, using two macros include in LPC802.h
I2C0->MSTTIME =
   I2C_MSTTIME_MSTSCLLOW(best_scl - 2u) | I2C_MSTTIME_MSTSCLHIGH(best_scl - 2u);
}
```

Set up the Timer (WKT) (part 1 of 3): *with LPO*

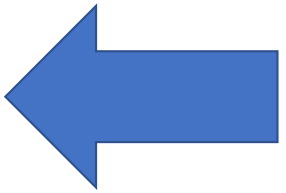
```
// Function name: WKT_Config
// Description:   Initialize a GPIO output pin and the WKT, then start it.
// Parameters:    None
// Returns:       Void
// Source: Ch. 18 of the User Manual (UM11045)
void WKT_Config()
{
    // ------------------------------------------------------------
    // Step 1: turn on the Wake-up Timer (WKT) enabling clock.
    SYSCON->SYSAHBCLKCTRL0 |= (SYSCON_SYSAHBCLKCTRL0_WKT_MASK);

    // ------------------------------------------------------------
    // Step 2: disable the vector lookup for WKT.
    NVIC_DisableIRQ(WKT_IRQn); // turn off the WKT interrupt.

    // ------------------------------------------------------------
    // Step 3: Turn on the Low Power Oscillator's power supply.
    // bit 6 to 0 in order to turn it ON. (Active low)
    SYSCON->                                SYSCON_PDRUNCFG_LPOSC_PD_MASK);
```

Fill in here



Set up the Timer (WKT) (part 2 of 3): *1 MHz LPO*

Fill in here



```
 ------------------------------------------------------------
// Step 4: Enable the LPOSC clock to the WKT.  (Bit 1 to 1)
SYSCON->                            |= (SYSCON_LPOSCCLKEN_WKT_MASK);


// ------------------------------------------------------------
// Step 5: Reset the WKT.
//       Set bit 9 to 0: assert (i.e. "make") the WKT reset
//  Set bit 9 to 1: clear the WKT reset (i.e. "remove" reset)
SYSCON->PRESETCTRL0 &= ~(SYSCON_PRESETCTRL0_WKT_RST_N_MASK);   // Reset the WKT
SYSCON->PRESETCTRL0 |= (SYSCON_PRESETCTRL0_WKT_RST_N_MASK);// clear the reset.
```

Set up the Timer (WKT) (part 3 of 3)

```
    // ------------------------------------------------------
    // Step 6: 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
}
```

Configure the WKT ISR

This will vary frequency based on LM75 temp

Fill in here

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


  // Toggle the LED
  GPIO->NOT[0] = (1UL<<LED_USER1);


  return;  // return nothing.
}
```

This global variable is modified during SysTick ISR