

ECEN 5023-001, -001B, -740

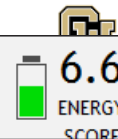
Mobile Computing & IoT Security

Lecture #10

16 February 2017



Assignment2 GNU ARM v4.8.3 - Debug (prof)



Session Counter	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode	Save	Reset	Compare
	974.54 nA	3.36 mW	60.09 nJ	26.07 s	Primary AEM Channel	Paused	↓	↺	↻
Selected Range	120.19 μ A	397.96 μ W	1.55 μ J	3.86 ms			×		



Assignment #2 review

- Integer Math accuracy issue
 - If the LFXO to ULFRCO is not an integer, due to the small number of clock ticks between excitation and measurement, an error occurs
 - Example:
 - OSC_Ratio = 1.2
 - Calibrated_Excite ticks = $4 * 1.2 = 4.8$
 - Calibrated_Excite ticks = ~~4.8~~ since clock ticks can only be integers
 - The answer is 5; must round up to insure minimum timing
 - To check for this truncation problem, a compare to float and integer is required
 - If (calibrated_float_ticks > calibrated_int_ticks) calibrated_int_ticks++

Assignment2 GNU ARM v4.8.3 - Debug (prof)

6.6
ENERGY
SCORE

Session Counter	Avg Current	Avg Power	Total Energy	Time Span	Channel	Mode	Save	Reset	Compare
	898.86 nA	3.21 mW	19.79 nJ	19.86 s	Primary AEM Channel	Paused	↓	↺	↻
Selected Range	88.12 μ A	291.70 μ W	1.40 μ J	4.75 ms			×		

Avg Volts



Calibrated ULFRCO running
in EM3

Verified

Calculating the ADC Acquisition cycles and prescaler

- T_a = number of ADC clocks for acquisition
- Resolution = Number of ADC bits being samples
- OSR = Oversampling
- $(T_a + \text{Resolution}) * \text{OSR} * \text{SamplesPerSecond} = \text{ADC_Clocks_Freq}$
- $T_a / \text{ADC_Clocks_Freq} \geq \text{Acquisition_Time}$ (use = to solve 2 equations with 2 variables)
 - $\text{ADC_Clocks_Freq} = T_a / \text{Acquisition_Time}$
- Solve for T_a
 - $(T_a + \text{Resolution}) * \text{OSR} * \text{SamplePerSecond} = T_a / \text{Acquisition_Time}$
 - $T_a * \text{OSR} * \text{SamplePerSecond} * \text{Acquisition_Time} + \text{Resolution} * \text{OSR} * \text{SamplePerSecond} * \text{Acquisition_Time} = T_a$
 - $T_a(1 - \text{OSR} * \text{SamplePerSecond} * \text{Acquisition_Time}) = \text{Resolution} * \text{OSR} * \text{SamplePerSecond} * \text{Acquisition_Time}$
 - $T_a = \text{Resolution} * \text{OSR} * \text{SamplePerSecond} * \text{Acquisition_Time} / (1 - \text{OSR} * \text{SamplePerSecond} * \text{Acquisition_Time})$

Calculating the ADC Acquisition cycles and prescaler

- Assignment: OSR = 1, no over sampling, 12 bit resolution, 20,000 samples per second, minimum acquisition time of 3uS
 - $T_a = \text{Resolution} * \text{OSR} * \text{SamplePerSecond} * \text{Acquisition_Time} / (1 - \text{OSR} * \text{SamplePerSecond} * \text{Acquisition_Time})$
 - $T_a = 12 * 1 * 20,000 * 3 \times 10^{-6} / (1 - 1 * 20,000 * 3 \times 10^{-6})$
 - $T_a = 0.72 / 0.94$
 - $T_a = 0.765$
 - Cannot have a partial clock, so must always round up, so $T_a = 1$
- Now, go back and calculate the ADC clock frequency to determine prescaler

Calculating the ADC Acquisition cycles and prescaler

- Calculating the prescaler
 - $(T_a + \text{Resolution}) * \text{OSR} * \text{SamplesPerSecond} = \text{ADC_Clocks_Freq}$
 - $(1 + 12) * 1 * 20,000 = \text{ADC_Clocks_Freq}$
 - $\text{ADC_Clock_Freq} = 260,000$
 - $\text{Prescaler} = \text{CPU_Freq} / \text{ADC_Clock_Freq} = 14,000,000 / 260,000$
 - $\text{Prescaler} = 53.8$
 - Prescalers need to be integers, round down to obtain desired ADC_Clocks_Freq or greater. Rounding down will give you too few ADC clocks
- Verify
 - $14,000,000 / 53 = 264,151$ ADC clocks which provides the required 260,000
 - Try rounding up to 54
 - $14,000,000 / 54 = 259,259$ ADC clocks, no meeting the requirement of 260,000 clocks
 - $T_a = 1 * 1/\text{ADC_Clock_Freq} = 1 * 1/260,000 = 3.85\mu\text{s}$ which is greater than $3\mu\text{s}$

Agenda

- Class Announcements
- I2C peripheral
- Reducing ADC Energy using DMA
- I2C Ambient Light Sensor and LPM Assignment

Class Announcements

- Quiz #5 is due at 11:59pm on February 19th, 2017 at 11:59pm
- I2C Ambient Light Sensor and LPM is due on Wednesday, February 22nd, 2017 at 11:59pm
- “The Embedded Club” is having its first meeting today, Friday the 17th at 5:00pm in ECCR 151
 - Conducting a workshop on "Linux command line and using VI"

Reducing ADC energy with DMA

Total points for this exercise is 10 points

6.0 pts for the questions

4.0 pts of the code

Questions are worth a total of 6.0pts

- Question 1: 5.0 – 5.2 Energy Score (0.2 pt), average off LED current $\leq 1.4\mu\text{A}$ (0.4pts), average one period current at 1s/div between 17.50 – 21.0 μA (0.4 pts)
- Question 2: 5.1 – 5.3 Energy Score (0.2 pt), average off LED current $\leq 1.4\mu\text{A}$ (0.4pts), average one period current at 1s/div between 15.25 – 17.5 μA (0.4 pts)
- Question 3: 36.5 to 38.5 mS
- Question 4: 5.0 – 5.2 Energy Score (0.2 pt), average off LED current $\leq 1.0\mu\text{A}$ (0.4pts), average one period current at 1s/div between 16.0 – 19.0 μA (0.4 pts)
- Question 5: 5.1 – 5.3 Energy Score (0.2 pt), average off LED current $\leq 1.0\mu\text{A}$ (0.4pts), average one period current at 1s/div between 12.5 – 15.0 μA (0.4 pts)
- Yes (0.25 pts)
- Yes (0.25 pts)
- Yes (0.25 pts)
- Yes (0.25 pts)

Reducing ADC energy with DMA

Code is worth 5.0 pts

- 1 pt total for the following defined statements:
- 0.5 points if code used #defined statements for the following:

```
#define ADC0_DMA_Channel      5
#define ADC0_DMA_Arbitration  dmaArbitrate1
#define TEMP_Sensor_DMA      true
```

- 0.5 points if code used 6 of 9 below #defined statements for the following or similar (they could be doing the shifting in the program):

```
#define Temp_Sense_Rate      20000
#define ADC0_resolution     adcRes12Bit
#define ADC0_reference      adcRef1V25
#define ADC0_channel        adcSingleInpTemp
#define ADC0_acq_time       adcAcqTime1
#define ADC0_rep            true
#define ADC0_warmup         adcWarmupNormal
#define LowerLimitInC      15.0    // In degrees C
#define UpperLimitInC      35.0    // In degrees C
```

- 0.5 points if there is a comment disclaiming the origins of the temperature routines and specifies that the particular temperature routine is covered by this IP statement/comment.

Reducing ADC energy with DMA

Code is worth 5.0 pts

- e. 0.5 pts if the ADC_Start command is after the DMA_ActivateBasic setup
- f. 0.5 point if the code works which is defined as below:
 - a. Using the cold spray, LED1 turns on when in "Light," (LED0 is Off)
 - b. Using the cold spray, LED1 turns on when in "Darkness," (LED0 is ON)
 - c. Darkness is indicated when by LED0 turning ON when temp is out of limit (LED1 ON)
 - d. Lightness is detected by turning of LED0 when temp is out of limits (LED1 ON) in EM3
- g. 0.5 points if the variable used to sum the 750 conversions is an int or unsigned int variable (none-floating point) to obtain the average for the temperature conversion
- h. 1.0 point if the result of summing the 750 conversions and going through the temperature routine outputs what appears to be a valid temperature. This will need to be done in the debugger.
- i. (-0.5 pts deduction) if the measurement between LED_Excite SET and LED_Sense measured is less than 4mS
- j. (-0.5pts deduction) if there is not comment attributing the sleep routines to Silicon Labs or if it does not mention which routines are covered by it.
- k. (-1.0pt deduction) if there is no #define switch to turn on and off the ADC DMA
- l. (-1.0pt deduction) if the ADC does not measure between 36.5 – 38.5mS

If the student answered yes to question 5 - 8, that the LEDs turned on and off properly while running in EM3, deduct points if while running their program, the LEDs did not turn on and off properly.

NOTE: Please forward name of students who got all the questions correct and tested out successfully to test for Professor bonus.

I2C Ambient Light Sensor and LPM Assignment

Objective: Develop I2C code for the Leopard Gecko as a master on the I2C bus. The I2C device is the TSL2651 ambient light sensor. For this assignment, the passive on board ambient light sensor will not be used, but the active TSL2651 will be used instead. The LETIMER0 will be used to determine when the TSL2651 should be load power management turned on and when it should be load power management disabled.

Due: Wednesday, February 22nd, at 11:59pm

Instructions:

1. Make any corrections to assignment #1 that are necessary
2. Make any corrections to assignment #2 that are necessary
3. Make any corrections to assignment #3 that are necessary
4. Create a #define to enable and disable the onboard passive light sensor
 1. For this assignment, you will disable the onboard passive light sensor

I2C Ambient Light Sensor and LPM Assignment

5. Determine the I2C1 pins on the Leopard Gecko expansion port to be used for the I2C port to the TSL2651 and use the following additional expansion port pins
 1. GPIO power pin for TSL2651: PD0
 2. TSL2651 Interrupt: PD1
 3. No I2C DMA is required for this assignment
 4. I2C1 SCL: PC5
 5. I2C1 SDA: PC4
6. Develop a GPIO interrupt service routine to handle the TSL2651 interrupts
7. Develop a Load Power Management routines for the TSL2651
 1. Power up the TSL2651 in the correct sequence
 2. Disable/power down the TSL2651 in the correct sequence
8. Develop a routine to initialize / program the TSL2651 after power on reset
 1. Threshold Low register to be set to 0x000f
 2. Threshold High register to be set to 0x0800
 3. Persistence to be set to 4
 4. Integration timing to be set to 101mS
 5. Low Gain

I2C Ambient Light Sensor and LPM Assignment

9. The LETIMER0 will now be configured as follows:
 1. Period extended to 4.25 seconds
 2. 3 LETIMER0 periods will be broken down into the following and repeat
 1. Period 1: Enable the TSL2651 onto the I2C bus, initialize / start the TSL2651, and monitor the TSL2651 via its interrupt line
 2. Period 2: Monitor the TSL2651 via its interrupt line
 3. Period 3: Disable the TSL2651
10. Upon receiving an interrupt from the TSL2651 on low light, LED0 is latched ON
 1. The LED0 remains latched on even if the TSL2651 is disabled
11. Upon receiving an interrupt from the TSL2651 on high light, LED0 is latched OFF
 1. The LED0 remains latched off even if the TSL2651 is disabled
12. The ADC0 temp sensor will need to be programmed to the following setting for this assignment:
 1. Conversions per second: 40,000
 2. Number of conversions: 1,000
 3. Utilizing DMA

I2C Ambient Light Sensor and LPM Assignment

13. #defined statements (additional)

1. Enabling the use of the TSL2651
2. #define of the key I2C parameters to set up the I2C Leopard Gecko including the I2C EM setting for the block and unblock routines
3. #define of the I2C address, control register addresses, and key settings of the TSL2651 such as the setting for low and high light
4. #define of the GPIO pins used for the TSL2651

14. Use the following pins for I2C Ambient Light Sensor and include them in the #defines

1. GPIO power pin for TSL2651: PD0
2. TSL2651 Interrupt: PD1

15. **NOTE:** All average currents should be taken at a time scale of 1s/div.

I2C Ambient Light Sensor and LPM Assignment

13. #defined statements (additional)

1. Enabling the use of the TSL2651
2. #define of the key I2C parameters to set up the I2C Leopard Gecko including the I2C EM setting for the block and unblock routines
3. #define of the I2C address, control register addresses, and key settings of the TSL2651 such as the setting for low and high light
4. #define of the GPIO pins used for the TSL2651

14. Use the following pins for I2C Ambient Light Sensor and include them in the #defines

1. GPIO power pin for TSL2651: PD0
2. TSL2651 Interrupt: PD1

15. **NOTE:** All average currents should be taken at a time scale of 1s/div.

I2C Ambient Light Sensor and LPM Assignment

Questions:

1. In a separate document to be placed in the drop box with the program code, please answer the following questions:
2. Running in **EM2**, and no darkness detected or outside temperature limits detected (no LEDs ON), after 60 seconds after resetting the session counters, what is the Energy Score while LETIMER minimum energy mode is set to **EM2** and using ADC DMA?
3. With self-calibration enabled in **EM3**, and no darkness or outside temperature limits detected (no LEDs on), after 60 seconds after resetting the session counters, what is the Energy Score while LETIMER minimum energy mode is set to **EM3** and using ADC DMA?
4. With self-calibration enabled in **EM3**, what is the average current in the LETIMER0 period three after the TSL2651 has been disabled and the MCU is back in **EM3** when all the LEDs are turned OFF?

I2C Ambient Light Sensor and LPM Assignment


5. With self-calibration enabled in EM3, what is the average current in the LETIMER0 period one after the TSL2651 has been enabled and the MCU is back in EM3 when all the LEDs are turned OFF?
6. Does the ADC0 temperature sensor still detect temperatures outside the set limits and turn on LED1?
7. Does the TSL2651 turn on the LED0 when darkness is detected and does it stay latched until lightness has been detected?

Deliverables:

1. One document that provides the answers to Assignment #4
2. Another document that contains your .c project source and .h files with LETIMER0 set to EM3 mode

BONUS: If you can beat the professor's Energy Score on question 2 running on the same STK3600 dev kit, you will get 1 bonus points. The assignment must also be turned in on time.

Assignment #4



[SHOP](#)[LEARN](#)[AVC](#)[FORUM](#)[DATA](#)

[START A PROJECT](#)[PRODUCTS](#)[BLOG](#)[TUTORIALS](#)[VIDEOS](#)[WISH LISTS](#)[DISTRIBUTORS](#)[SUPPORT](#)

New Products

🔥 Top Sellers

SparkFun Originals

Sale

Gift Certificates

Arduino +

Audio

Books

Breakout Boards

Cables +

Components +

Development Tools +

Dings and Dents

Educators

GPS +

Intel® Edison

IoT +

Kits

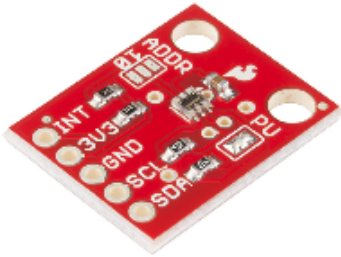
LCDs +


Prototyping +


Raspberry Pi

Robotics +

HOME / PRODUCT CATEGORIES / LIGHT / IMAGING / SPARKFUN LUMINOSITY SENSOR BREAKOUT - TSL2561







© images are [CC BY-NC-SA 3.0](#)

3D Download: [Sketchup](#), [STL](#), [Blender](#)

SparkFun Luminosity Sensor Breakout - TSL2561 project on

SparkFun Luminosity Sensor Breakout - TSL2561

SEN-12055 ROHS ✓ ✨

★★★★★ 1

Description: The TSL2561 SparkFun Luminosity Sensor Breakout is a sophisticated light sensor which has a flat response across most of the visible spectrum. Unlike simpler sensors, the TSL2561 measures both infrared and visible light to better approximate the response of the human eye. And because the TSL2561 is an integrating sensor (it soaks up light for a predetermined amount of time), it is capable of measuring both small and large amounts of light by changing the integration time.

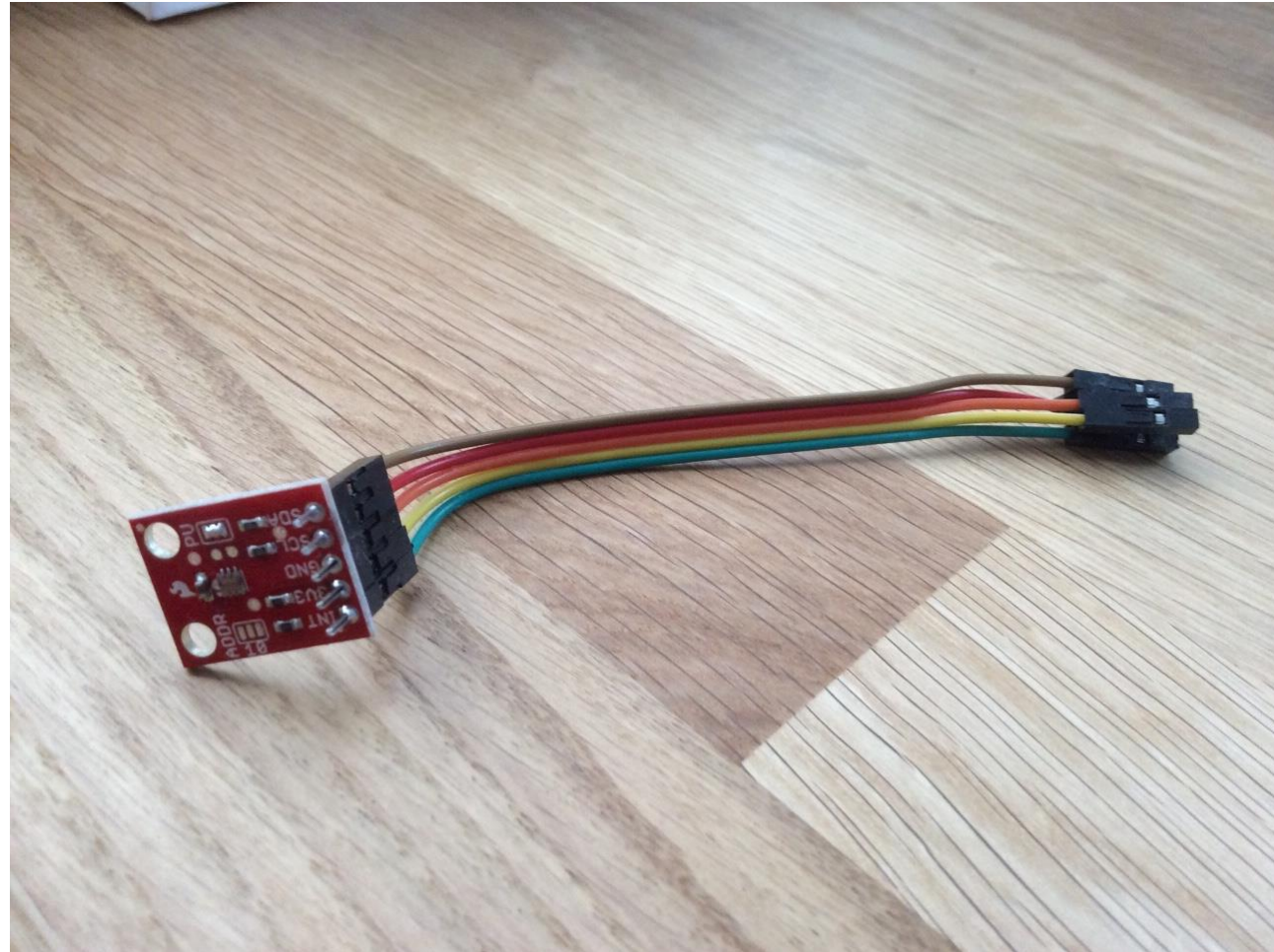
The TSL2561 is capable of direct I2C communication and is able to conduct specific light ranges from 0.1 - 40k+ Lux easily. Additionally, the TSL2561 contains two integrating analog-to-digital converters (ADC) that integrate currents from two photodiodes, simultaneously. Each breakout requires a supply voltage of 3V and a low supply current max of 0.6mA.

Documents:

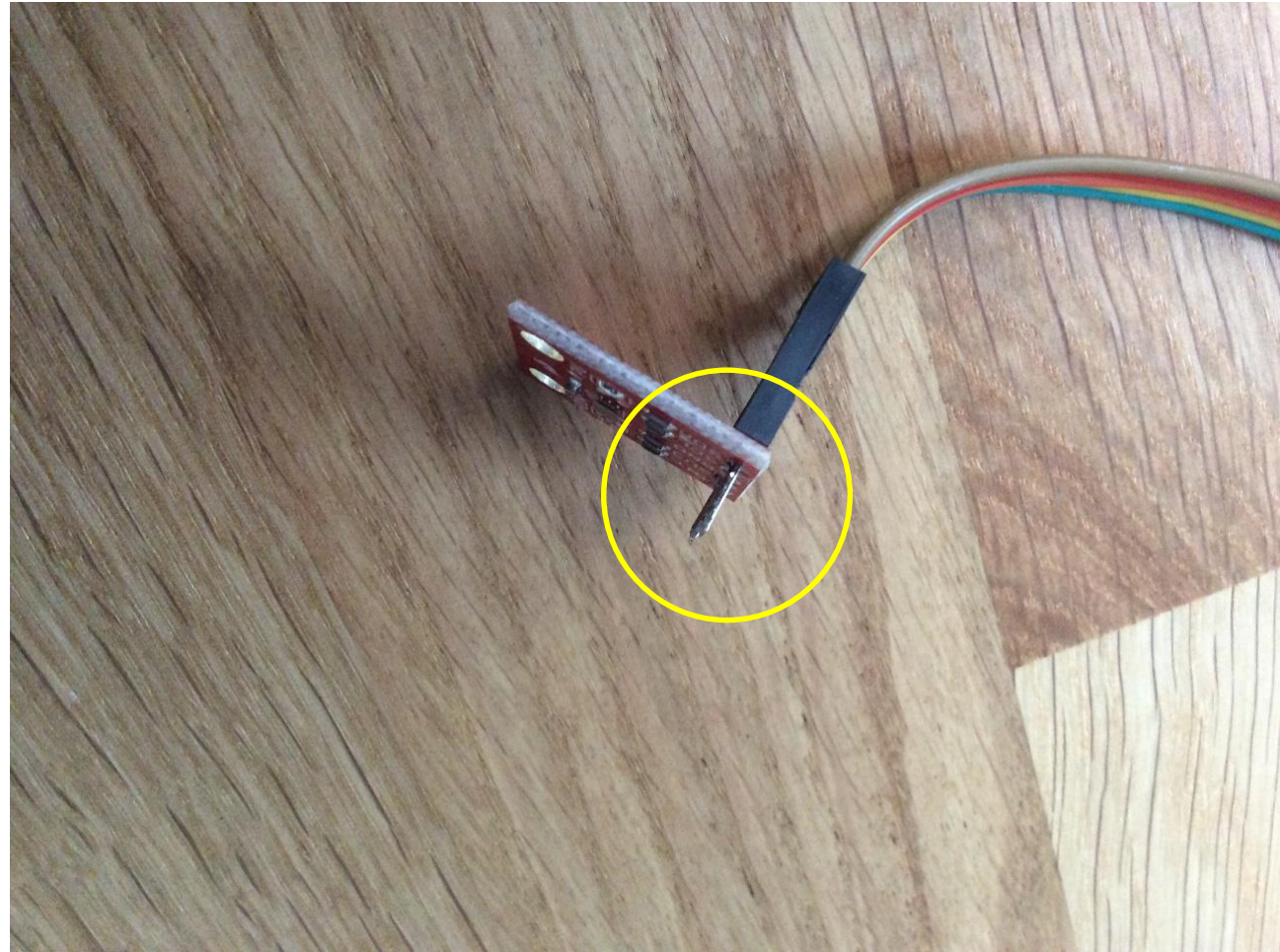
- [Schematic](#)
- [Eagle Files](#)
- [Datasheet](#)
- [Hookup Guide](#)
- [GitHub](#) (Design Files & Example Code)
- [GitHub](#) (Library)
- [Product Video](#)



I2C Ambient Light Sensor and LPM Assignment



I2C Ambient Light Sensor and LPM Assignment



Example of what is expected of the I2C header files for the TSL2651

```
/* Accelerometer definitions */
```

```
#define Acc_Int1_Port    gpioPortD
#define Acc_Int1_Pin     6U
#define Acc_Int1_Rising_Int  false
#define Acc_Int1_Falling_Int true
```

```
/* Accelerometer interrupt */
```

```
/* Accelerometer register definitions */
```

```
#define Acc_Chip_Addr    0x3A
#define Acc_I2c_Port     1
```

```
#define Acc_Who_Am_I_Addr 0x0d
```

```
#define Acc_Who_Am_I     0x2A
```

```
#define Acc_Data_Base_Reg 0x00
```

```
#define Acc_Num_Data_Regs 0x07
```

```
#define Acc_Sysmod_Addr   0x0B
```

```
#define Acc_Sysmod        (0x03 << 0)
```

```
#define Acc_Int_Source_Addr 0x0c
```

```
#define Acc_Src_Aslp       (0x01 << 7)
```

```
#define Acc_Src_Trans       (0x01 << 5)
```

```
#define Acc_Src_Lndprt      (0x01 << 4)
```

```
#define Acc_Src_Pulse       (0x01 << 3)
```

```
#define Acc_Src_Ff_Mt       (0x01 << 2)
```

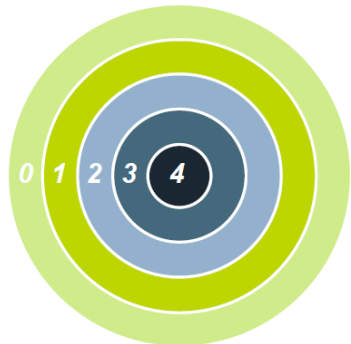
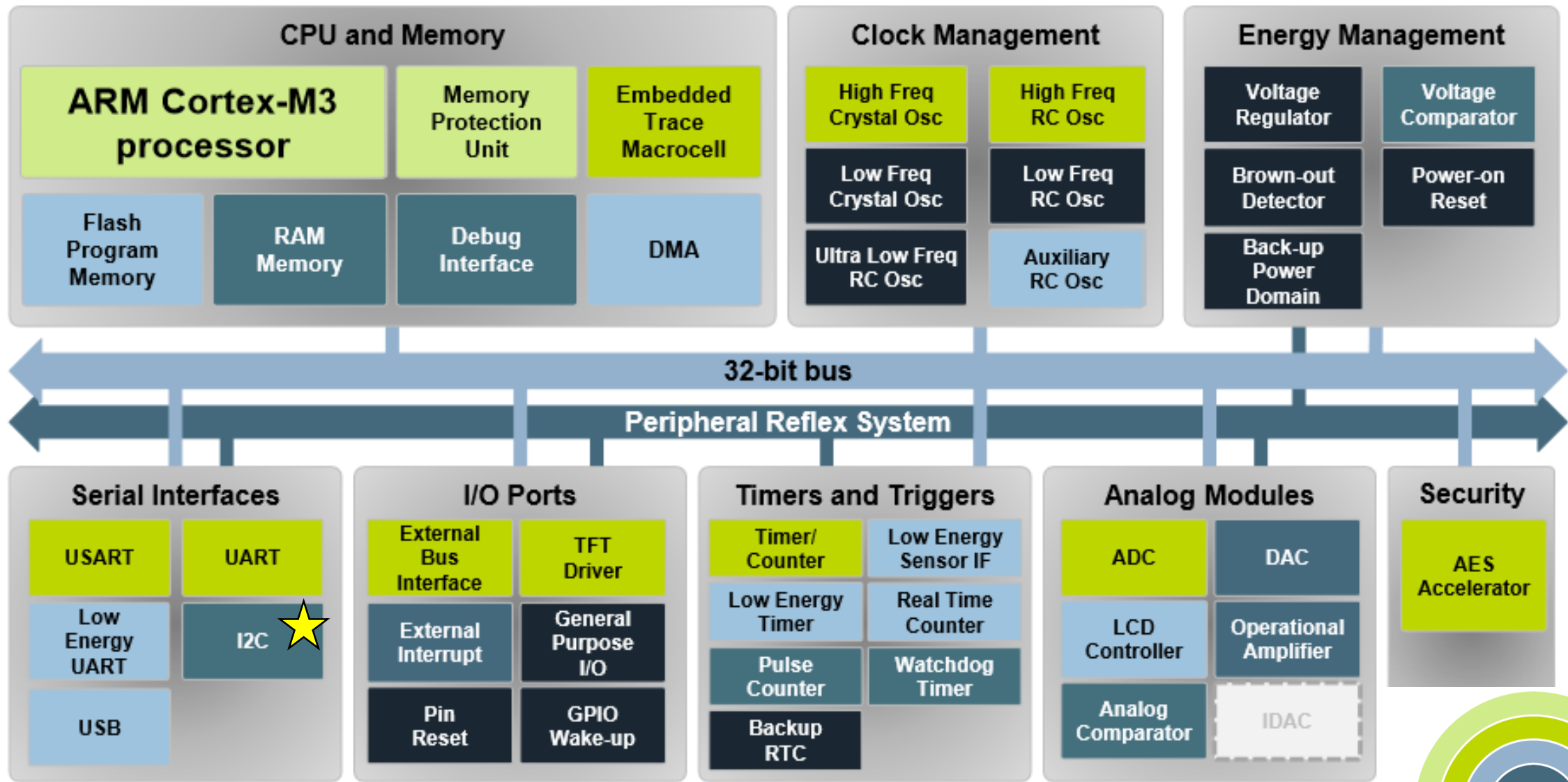
```
#define Acc_Src_Drdy        (0x01 << 0)
```

```
#define Acc_Xyz_Data_Cfg_Addr 0x0e
```

```
#define Acc_Hpf_Out          (0x00 << 4)    // Disable high pass filter data
```

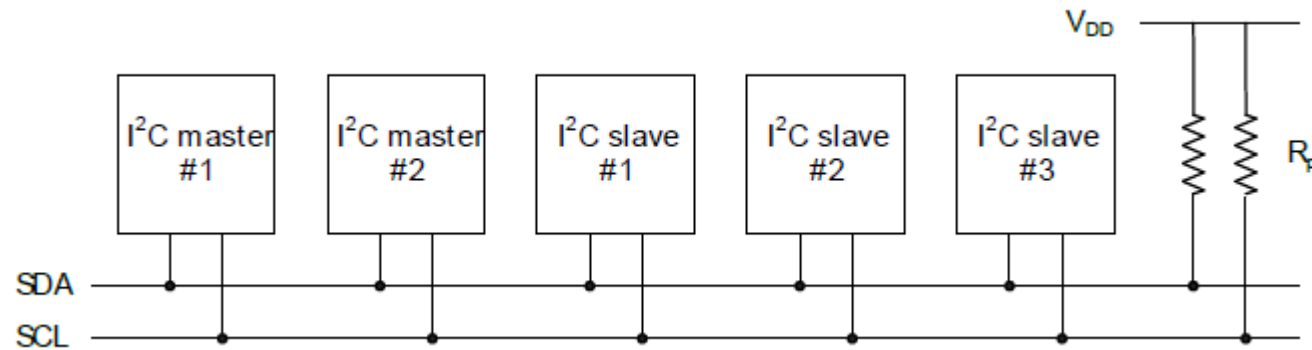
```
#define Acc_Fs                (0x02 << 0)    // Set full range to be 8G
```

```
#define Acc_Xyz_Data_Cfg     Acc_Hpf_Out | Acc_Fs
```



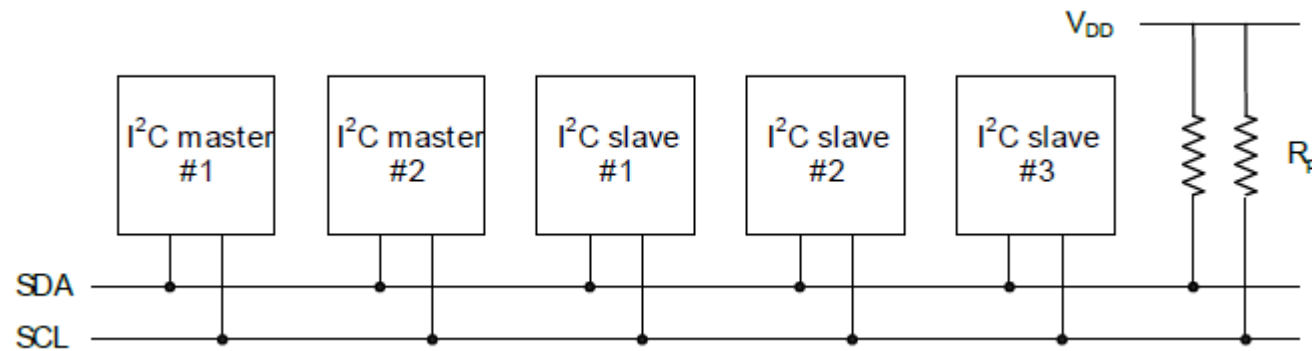
What is I2C?

- The I2C-bus uses two wires for communication
 - A serial data line (SDA)
 - A serial clock line (SCL)
- It is a true multi-master bus that includes collision detection
- Arbitration to resolve situations where multiple masters transmit data at the same time without data loss.

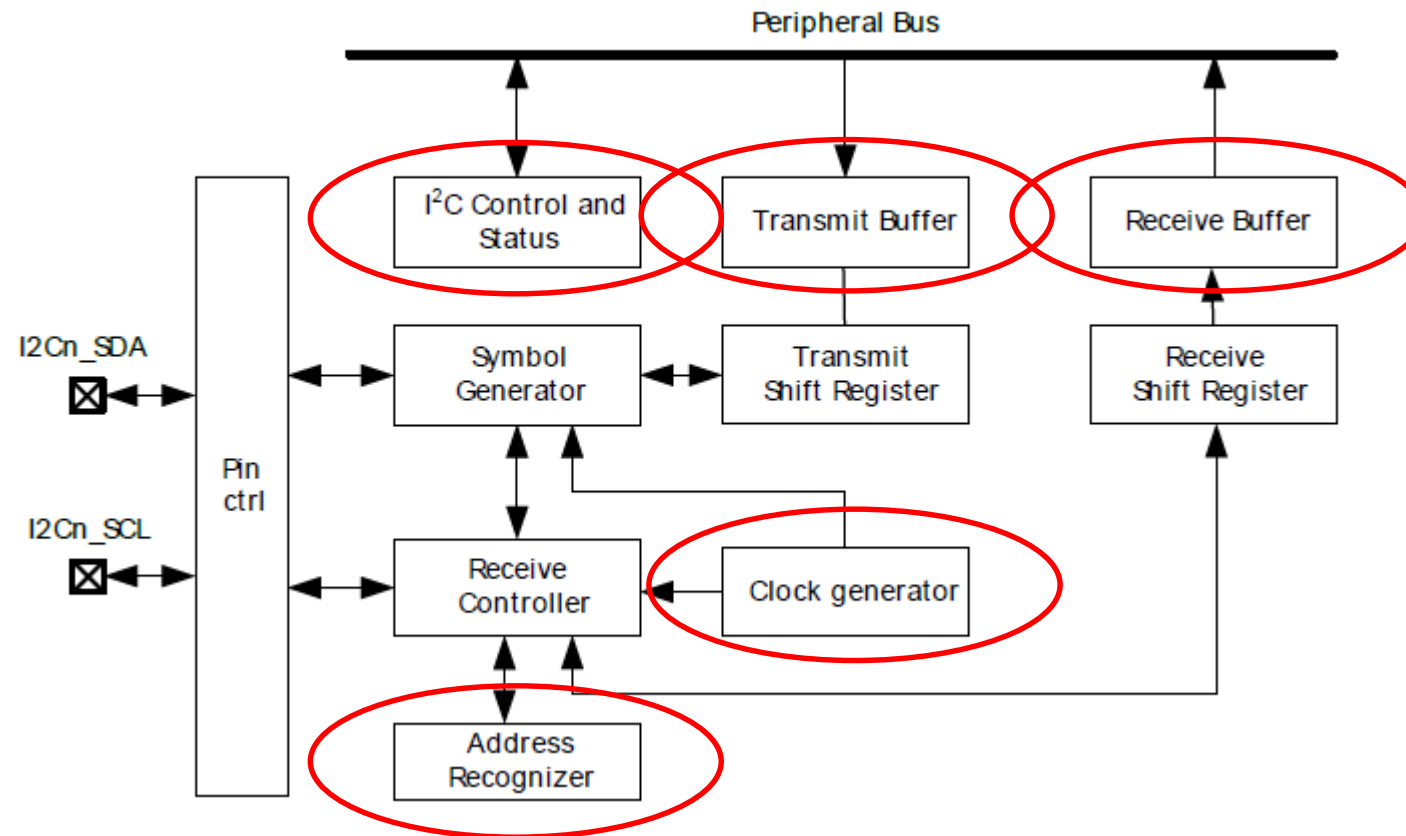


What is I2C?

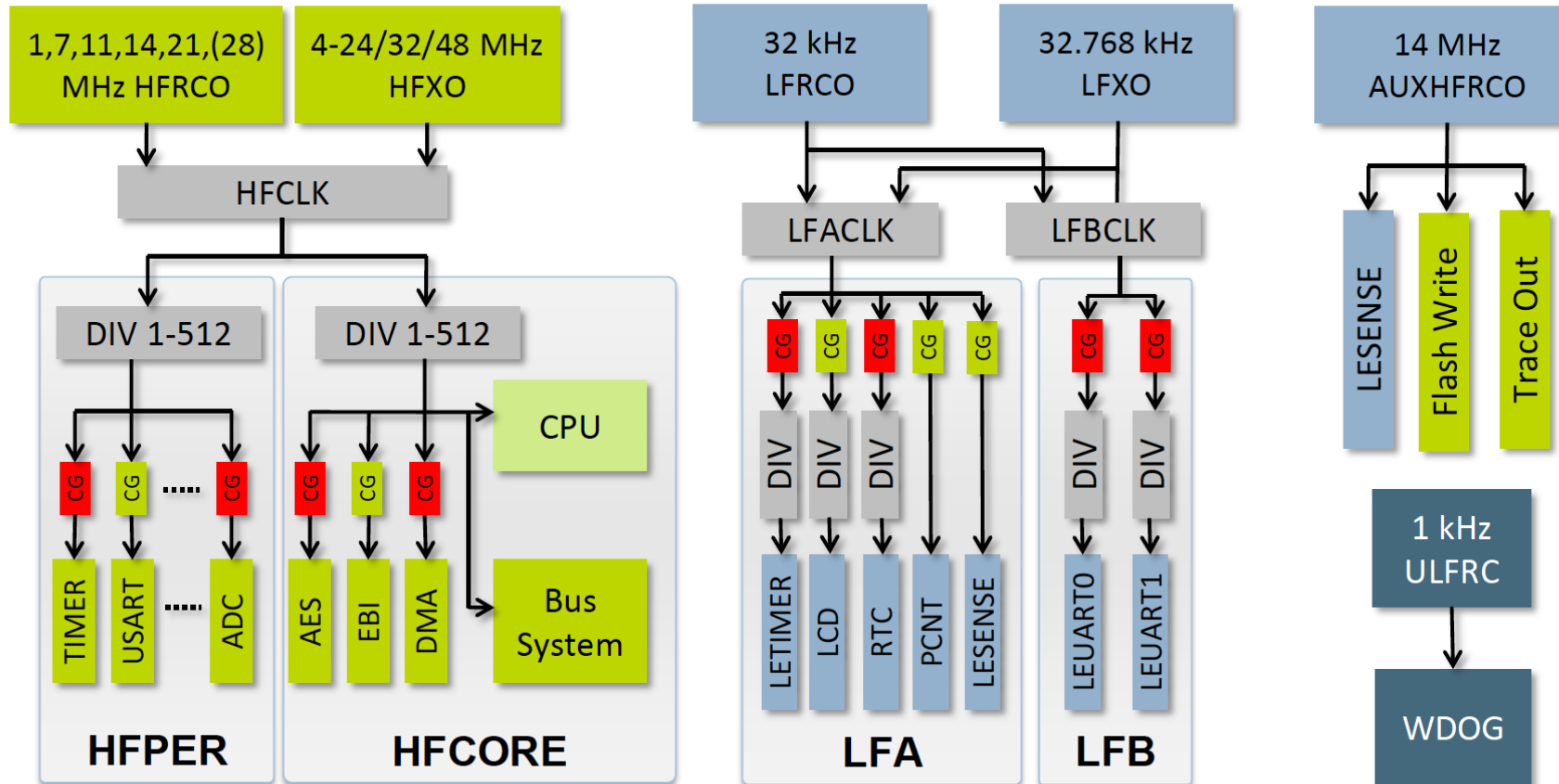
- Each device on the bus is addressable by a unique address
- The I2C master can address all the devices on the bus, including other masters
- Both the bus lines are open-drain. The maximum value of the pull-up resistor can be calculated as a function of the maximal rise-time t_r for the given bus speed
- The maximal rise times for 100 kHz, 400 kHz and 1 MHz I2C are 1 μ s, 300 ns and 120 ns respectively.



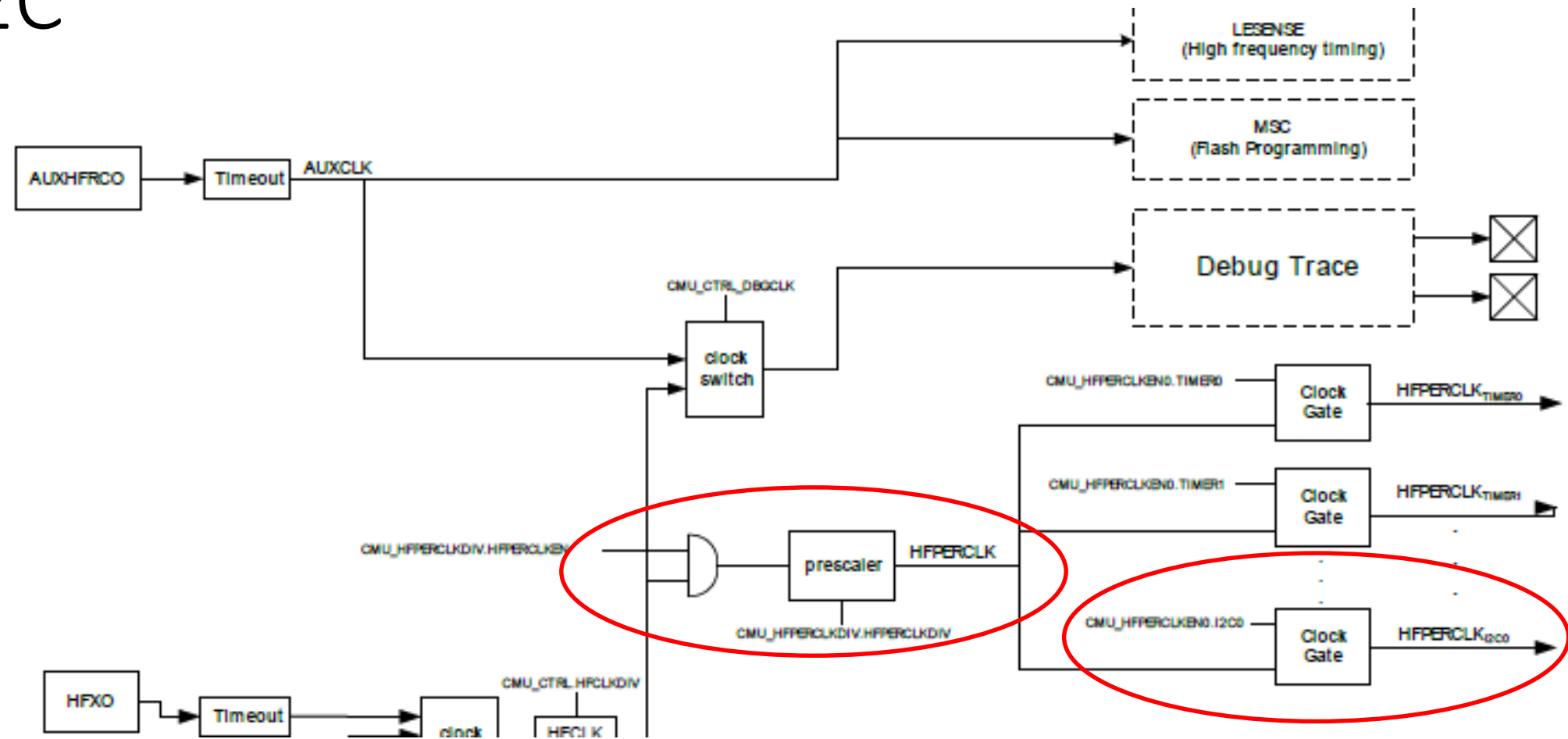
I2Cn peripheral block diagram



Clocks and Oscillators



I2C

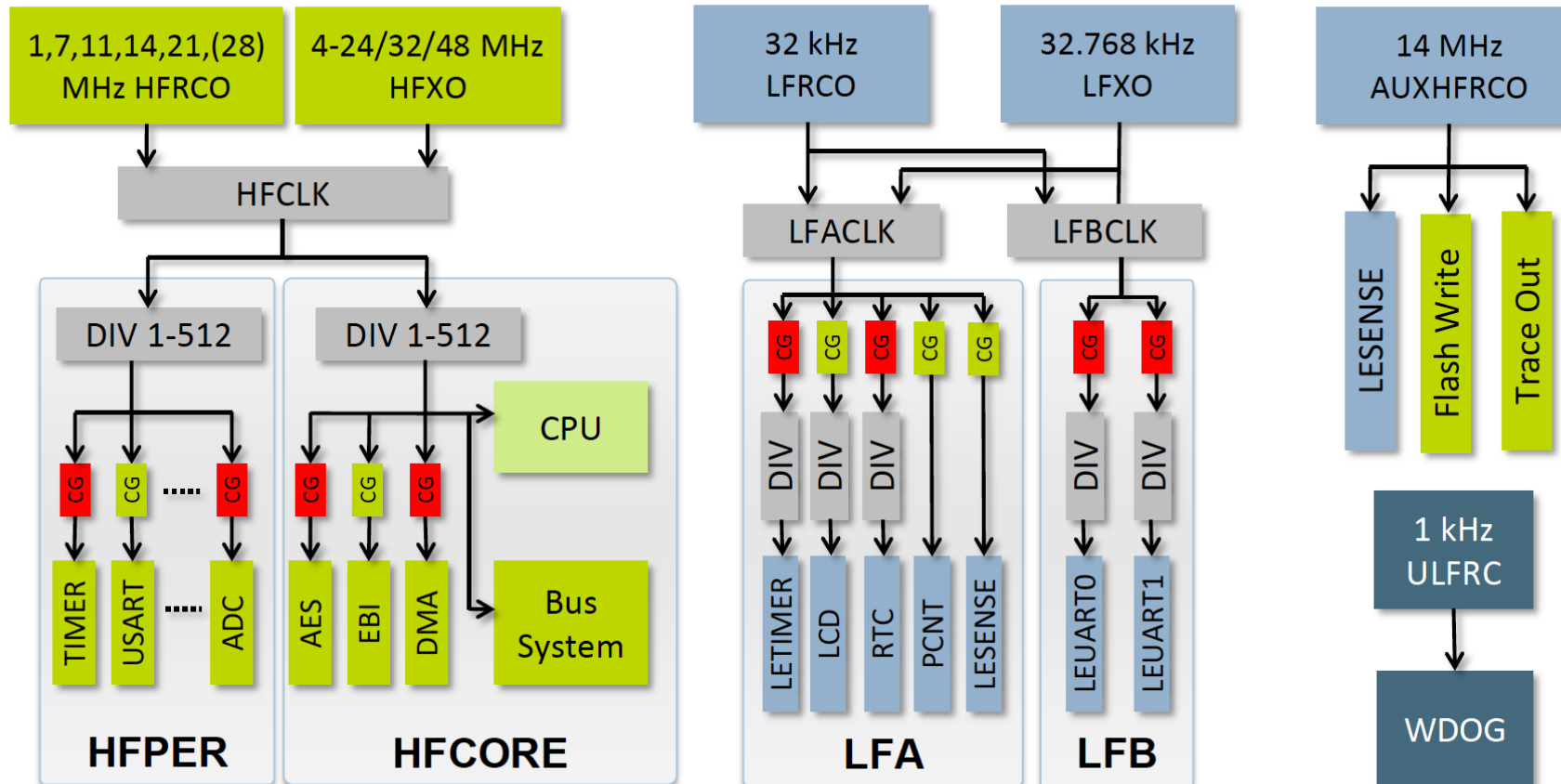


Clocks and Oscillators



I2C is connected to the HFPERCLK, which is always running in EMO & EM1, so what does this mean?

You will need to enable the HFPERCLK



Will need to enable the clock to the I2C

Setting up the I2C

- First, the clock tree to the I2C must be established
 - Without establishing the clock tree, all writes to the I2Cn registers will not occur
 - I2Cn clock source is the HFPERCLK, so no oscillator enable is required, but the HFPERCLK needs to be enabled using [CMU_ClockEnable](#)
 - Pseudo code in the CMU setup routine to enable the I2Cn clock tree:
 - Lastly, enable the I2C clocking using the [CMU_ClockEnable](#) for the I2Cn

Setting up the I2C

- Second, the I2C must be set up
 - Specify SCL and SDA pins of the I2Cn peripheral
 - Recommend using the GPIO pin mode set up emlib routines
 - Silicon Labs' I2C application note, [AN0011](#), software examples is available resource to insure the GPIO pins are set up correctly

I2Cn routing information

EFM[®]32
...the world's most energy friendly microcontrollers

Alternate	LOCATION							Description
Functionality	0	1	2	3	4	5	6	
GPIO_EM4WU0	PA0							Pin can be used to wake the system up from EM4
GPIO_EM4WU1	PA6							Pin can be used to wake the system up from EM4
GPIO_EM4WU2	PC9							Pin can be used to wake the system up from EM4
GPIO_EM4WU3	PF1							Pin can be used to wake the system up from EM4
GPIO_EM4WU4	PF2							Pin can be used to wake the system up from EM4
GPIO_EM4WU5	PE13							Pin can be used to wake the system up from EM4
HFXTAL_N	PB14							High Frequency Crystal negative pin. Also used as external optional clock input pin.
HFXTAL_P	PB13							High Frequency Crystal positive pin.
I2C0_SCL	PA1	PD7	PC7	PD15	PC1	PF1	PE13	I2C0 Serial Clock Line input / output.
I2C0_SDA	PA0	PD6	PC6	PD14	PC0	PF0	PE12	I2C0 Serial Data input / output.
I2C1_SCL	PC5	PB12	PE1					I2C1 Serial Clock Line input / output.
I2C1_SDA	PC4	PB11	PE0					I2C1 Serial Data input / output.
								LCD voltage booster (optional) boost capacitor negative

Setting up the I2C

- Second, the I2C must be set up
 - Specify SCL and SDA pins of the I2Cn peripheral
 - Recommend using the GPIO pin mode set up emlib routines
 - Silicon Labs' I2C application note, AN0011, software examples is available resource to insure the GPIO pins are set up correctly
 - Must route the I2C pins to the I2Cn peripheral
 - This can be accomplished by writing the correct location register into `I2Cn->ROUTE`
 - Need to specify the I2C init Type_Def
 - `I2C_Init(I2Cn, &init_Type_Def);`

Setting up the I2C

- Third, I2Cn bus must be reset
 - Upon setting up the I2C bus, the bus and its peripherals may be out of synch
 - To reset the I2C bus, the following procedure should be executed:

```
/* Exit the busy state. The I2Cn will be in this state out of RESET */  
if (I2Cn->STATE & I2C_STATE_BUSY){  
    I2Cn->CMD = I2C_CMD_ABORT;  
}
```

Setting up the I2C

- Forth, the I2C interrupts must be enabled if needed
 - Clear all interrupts from the I2C to remove any interrupts that may have been set up inadvertently by accessing the `I2Cn->IFC` register or the emlib routine
 - Enable the desired interrupts by setting the appropriate bits in `I2Cn->IEN`
 - Set `BlockSleep` mode to the desired Energy Mode
 - The Leopard Gecko can be an I2C Master in EM0 & EM1
 - The Leopard Gecko can detect its I2C Slave address down into EM3 since the clock is generated from the I2C bus clock SCL
 - Enable interrupts to the CPU by enabling the I2Cn in the Nested Vector Interrupt Control register using `NVIC_EnableIRQ(I2Cn_IRQn);`

Setting up the I2C

- Fifth, the I2Cn interrupt handler must be included
 - Routine name must match the vector table name:
`Void I2Cn_IRQHandler(void) {
 }
}`
 - Inside this routine, you add the functionality that is desired for the I2Cn interrupts

Writing your own I2C driver

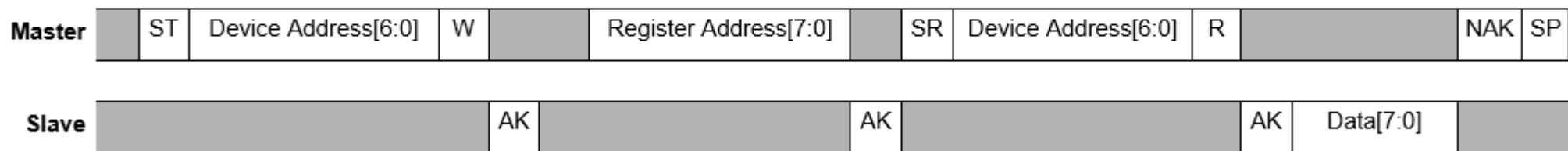
- The I2C standard appears to be more of a physical bus standard than a bus protocol
 - Bus protocol in this usage is a defined sequence of operations that could be taken from one device to another with simple port to the specific devices specifications
 - I have found that many I2C devices use the I2C physical bus protocol, but do not easily fit into a standard I2C library

Writing your own I2C driver

- Where to start?
 - Go to the I2C slave's data sheet and find their I2C bus sequence of events diagram

I²C data sequence diagrams

< Single-byte read >

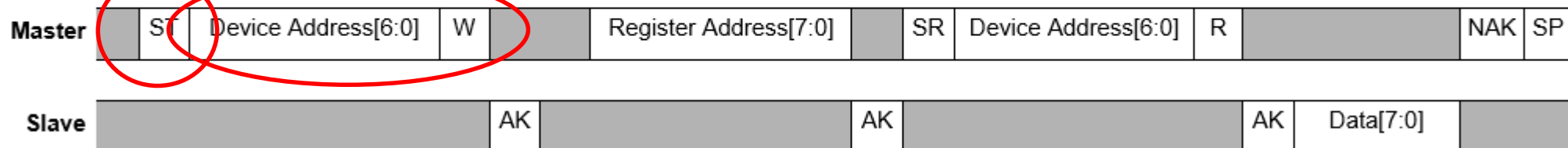


Writing your own I2C driver

- Now, convert the visual diagram into a driver
- Prime the TX Buffer for the Start Command
 - $I2Cn \rightarrow TXDATA = (I2C_device_addr \ll 1) \mid R/W \text{ bit} = 0$ signifying write of address to the slave;
- Now send the Start Bit
 - $I2Cn \rightarrow CMD = I2Cn_CMD_START;$

I²C data sequence diagrams

< Single-byte read >

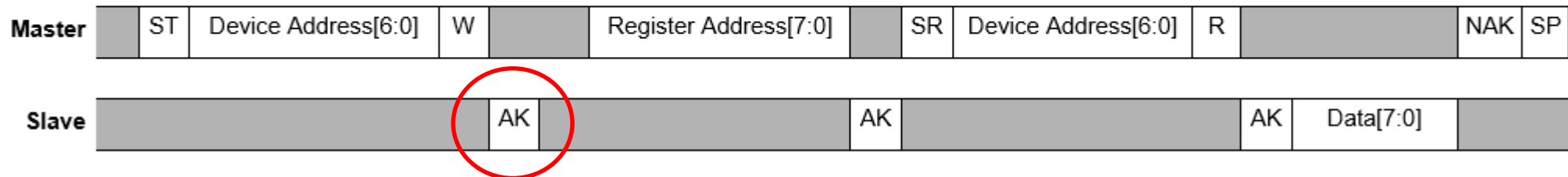


Writing your own I2C driver

- Now, wait for the slave to respond
 - While ((I2Cn->IF & I2Cn_IF_ACK) == 0);
 - After the ACK has been received, it must be cleared from the IF reg
 - I2Cn->IFC = I2Cn_IFC_ACK;

I²C data sequence diagrams

< Single-byte read >

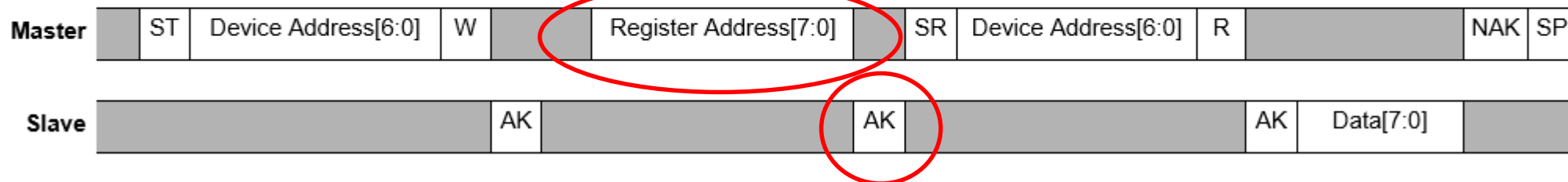


Writing your own I2C driver

- Now, send the I2C device register address
 - `I2C->TXDATA = I2C_device_reg_add;`
- Now, wait for the slave to respond
 - `While ((I2Cn->IF & I2Cn_IF_ACK) == 0);`
 - After the ACK has been received, it must be cleared from the IF reg
 - `I2Cn->IFC = I2Cn_IFC_ACK;`

I²C data sequence diagrams

< Single-byte read >

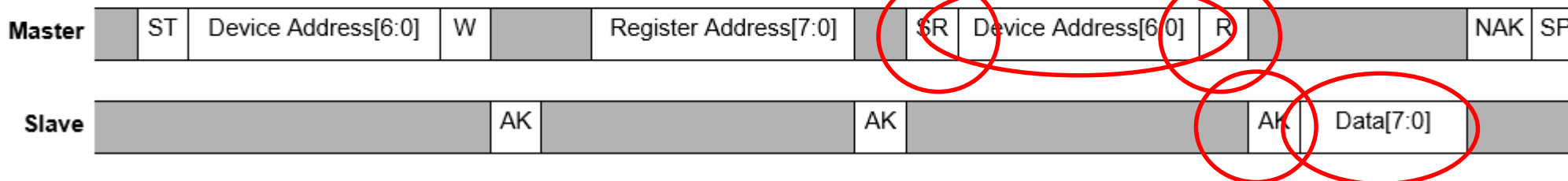


Writing your own I2C driver

- Your driver continues down through out the visual diagram
 - SR = Start Repeat
 - Device Address
 - R = Read/Write bit set to 1 for Read Operation
- Wait for slave to respond
 - Ak = Acknowledge
 - Data[7:0]

I²C data sequence diagrams

< Single-byte read >

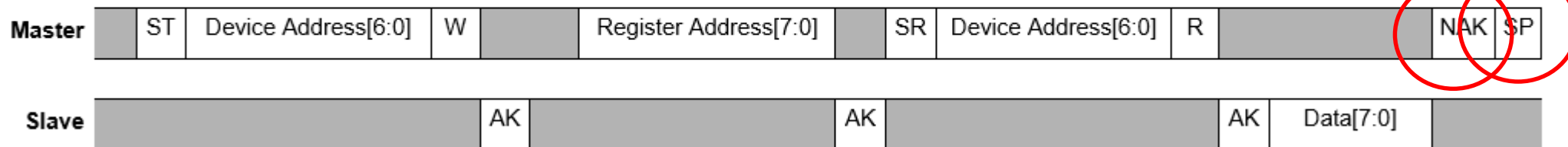


Writing your own I2C driver

- Finish transfer
 - NAK
 - SP – Stop bit

I²C data sequence diagrams

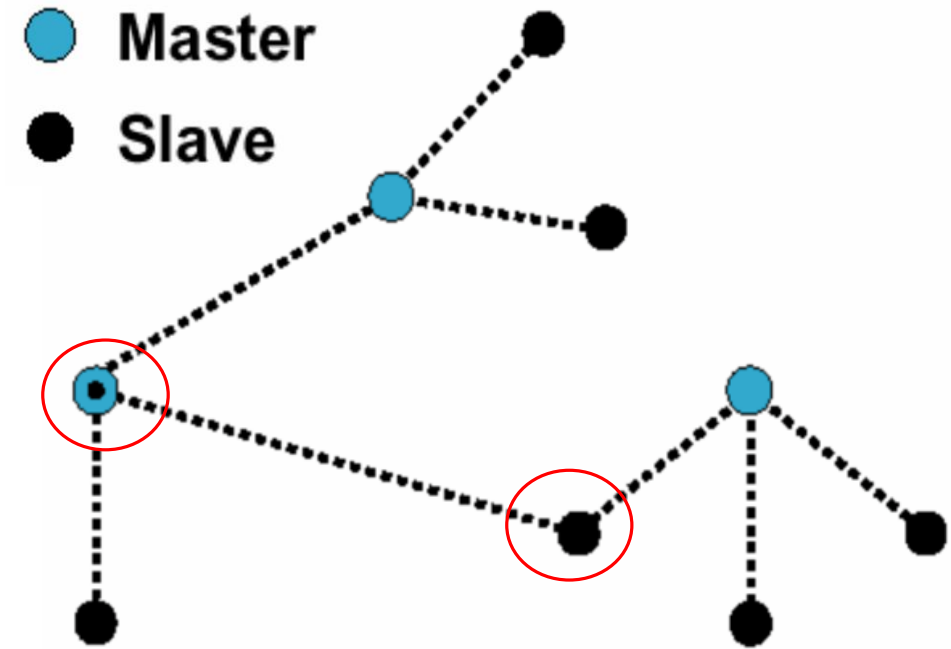
< Single-byte read >



Bluetooth Classic - Piconet-to-Piconet: The Scatternet



- Scatternets allow devices to be active in numerous piconets
- The device can be a slave in one piconet and a master in another. It **cannot** be a master in two piconets which would result in two Piconets with the same frequency hopping sequence.
- The device can act as a gateway from one piconet to another.
- Before a slave leaves to participate in another Piconet, the slave must **inform** the current master that it will be unavailable for a period of time.
- As soon as a master leaves a Piconet to participate in another Piconet, all traffic within the original Piconet is **suspended** until the master returns.



Bluetooth Classic – Identifying Bluetooth Devices



- Each Bluetooth device is assigned a unique 48-bit MAC address by the Bluetooth SIG
- This is enough addresses for 281,474,976,710,656 Bluetooth units, this should last a few years even with the optimistic predictions of the analysts!
- The address is split into three parts:
 - LAP: Lower Address Part - used to generate frequency hop pattern and header sync word.
 - UAP: Upper Address Part - used to initialize the HEC and 1sb CRC engines.

LAP [0:23]

UAP[24:31]

NAP [32:47]



Bluetooth Classic – Bluetooth Channels

- A master can create two types of logical channel with a slave device:
 - Asynchronous Connection Less (ACL): Packet Switched System provides a reliable data connection with a best effort bandwidth; depends on radio performance and number of devices in the piconet.
 - Synchronous Connection Oriented (SCO): Circuit Switched System provides real time reliable connection with a guaranteed bandwidth; usually used for voice based applications.
- The Bluetooth connections are limited to 1Mbps across the air
 - Giving a theoretical maximum of ~723kbps of useable data

Bluetooth Classic – What Bluetooth Is Not?

- Bluetooth is not intended to compete with or replace 802.11b, they are complimentary technologies
- The data rates, usage scenarios and fundamental ethos behind them are all different!
- It is unlikely to be used in corporate wireless LAN's. It is not suitable for high data rate applications
- High is defined to be >600kbps this allows suitable margin for re-transmissions
- Therefore, high quality video streaming is not possible.

Bluetooth Classic - The ISM Band

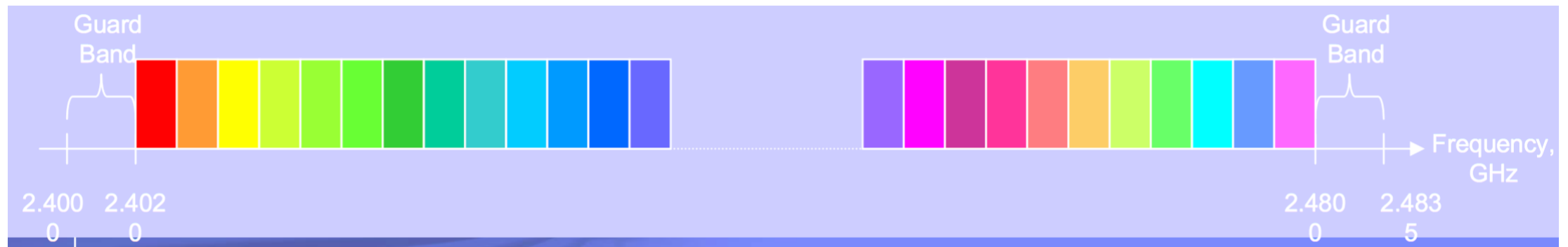
- Bluetooth uses the 2.4GHz ISM frequency band
- The Industrial, Scientific and Medical (ISM) band is an unlicensed band, i.e. any one can use it provided they don't exceed certain power constraints
- The 2.4GHz ISM band is unlicensed all over the world which makes Bluetooth the only completely world wide standard
- Bluetooth uses the frequency range 2.4000 - 2.4835GHz

Bluetooth Classic – Overcoming Interference

- Due to the unrestricted nature of the ISM band, Bluetooth must overcome interference from other systems and minimize its interference on other systems
- Bluetooth does this by using a Frequency Hopping Spread Spectrum (FHSS) technique
- This spreads the RF power across the spectrum which reduces interference and the spectral power density.

Bluetooth Classic - Frequency Hopping Spread Spectrum - FHSS

- Bluetooth splits the spectrum up into 79 1MHz wide channels with a small guard band at each end of the whole band
- The Bluetooth radio changes transmission frequency 1600 times a second
- The frequency hops follow a pseudo random sequence that meets the power density requirements for the FCC and other regulatory bodies



Bluetooth Classic - Hop Selection and Synchronization

- One frequency hop lasts 625us, this increment is called a time slot
- Each Bluetooth device has a clock circuit that counts frequency hops
- The address of the master of the piconet is used to seed a frequency hop calculation algorithm
- The phase of the hop sequence is defined by the Bluetooth clock of the master
- Device address and clock phase information is exchanged during connection negotiation
- The slave synchronizes its own clock to the master's during connection so that both devices change frequency at the same time

Bluetooth Classic – Transmission Timing

- A slave can only send data to the master after it has received a valid packet from the master
- Masters transmit in **even** numbered slots and slaves respond in the next **odd** numbered slot
- Single slot packets are less than 366 μ s long to allow the synthesizer to retune to the next frequency hop
- Multi-slot packets of 3-slot and 5-slot packets are possible for higher data rates.
 - During 3 and 5-slot transfers, the radio transmitters remain on the same frequency

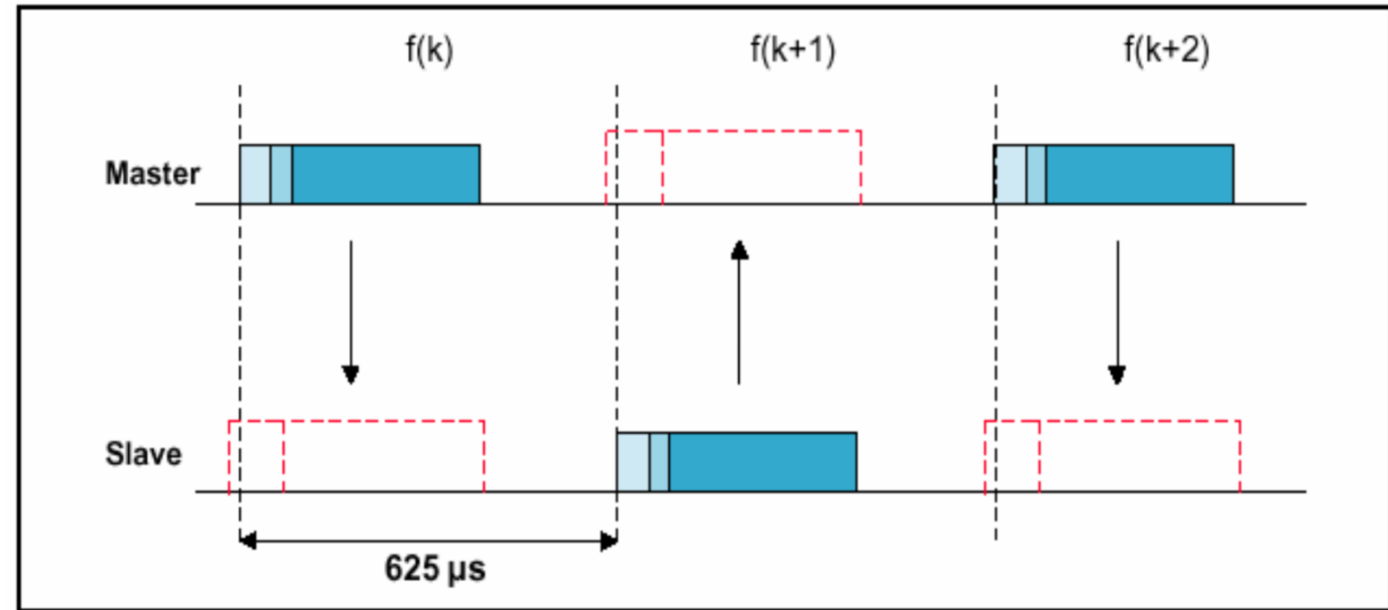


Figure 2.1: TDD and timing

Bluetooth Classic – Power Classes

- Bluetooth defines 3 power classes for devices:
 - Class 1: 0dBm to +20dBm (1mW to 100mW)
 - Class 2: -6dBm to +4dBm (250uW to 2.5mW)
 - Class 3: <0dBm (<250uW)
- These power classes translate in to approximate distances often used when discussing Bluetooth:
 - Class 1: 100 Meters
 - Class 2: 10 Meters
 - Class 3: <10 Meters

Bluetooth Classic - States

- **Standby** state: Each device which is currently not participating in a piconet (and not switched off) is in Standby mode.
 - This lower-power mode where only the device's native clock is running
- **Inquiry** state: A device wants to establish a Piconet or a device just wants to listen to see if something is going on.
 - **Establishing** a piconet – A device starts the inquiry procedure by sending an inquiry access code (IAC) that is common to all Bluetooth devices. The IAC is broadcast over 32 so-called wake-up carriers in turn.
 - A device in standby that listen periodically can wake up and respond to an IAC request by returning a packet address and timing information required by the master to initiate a connection. It is now a slave device. It receives an Active Member Address (AMA)
 - Note: There are 3-bits for Active Member Addresses which limit the Piconet to 8 total devices. There are 8-bits for Parked Member Addresses so there can be 64 parked devices associated with the piconet.

Bluetooth Classic - States

- **Page** state: Once an inquiry is successful, the device enters the page state.
 - In page state, the master set up connections to each device.
- **Connection** state:
 - Comprises of all active and low power devices in the piconet
 - **Active** state: Slaves participate in the piconet by listening, transmitting, and receiving.
 - **Low** power states:
 - **Sniff** state: Highest power consuming low power state. The device listens to the piconet at a reduced rate. Not on every other slot as a active device.
 - **Hold** state: The device does not release its Active Member Address, but stops all ACL transmissions.
 - **Park** state: Lowest of the power saving states. The device releases its AMA and receives a Parked Member Address (PMA). The device is still a member of the Piconet, but gives room to another device to be an active member.

Required book for
Bluetooth portion
of course

