ECEN 5823-001
Internet of Things Embedded Firmware

Lecture #6

13 September 2018

Agenda

- Class Announcements
- Managing Energy Modes Rubric
- I2C Load Management Assignment
- I2C Peripheral
- Writing your own I2C routine
- Scheduling
- Bluetooth Classic

Class Announcements

- Quiz #3 is due at 11:59 on Sunday, September 13th, 2018
- Homework #1: Managing Energy Modes is due on Saturday, September 15th, at 11:59pm
- Homework #2: I2C Load Management Assignment is due on Saturday, September 22nd, at 11:59pm

Letimer0 prescaler calculation

- What variables do you require to calculate the prescaler?
- Is the LETIMER0 prescaler linear or exponential to the power of 2?
- What formula to calculate the prescaler?
- What is the formula to calculate the COMP0 count?

ECEN 5023-001, -001B

Fall 2018

Managing Energy Modes Rubric

Managing Energy Modes Rubric

1. Total points for this exercise is 10 points
   a. 5.0 pts for the questions
   b. 5.0 pts of the code

2. Question scoring. Max score is 5.0 pts.
   a. Question 1: EM0
      i. Period average current: 4.5 – 5.2mA (0.4 pts)
      ii. Current LED off: 4.4 – 5.1mA (0.4 pts)
      iii. Current LED on: current in (ii) plus 0.40 to 0.55mA (0.2pts)
   b. Question 2: EM1
      i. Period average current: 3.0 – 3.9mA (0.4 pts)
      ii. Current LED off: 3.2 – 3.8ma (0.4 pts)
      iii. Current LED on: current in (ii) plus 0.40 to 0.55mA (0.2 pts)

I2C Load Management Assignment

ECEN 5823
Si7021 and Load Power Management
Fall 2018

)bjective: Adding the Si7021 temp/humidity via the I2C bus and enabling / disabling the Si7021 o implement load power management.

Note: This assignment will begin with the completed Managing Energy Mode Assignment

)ue: Saturday, September 22nd, 2018

nstructions:

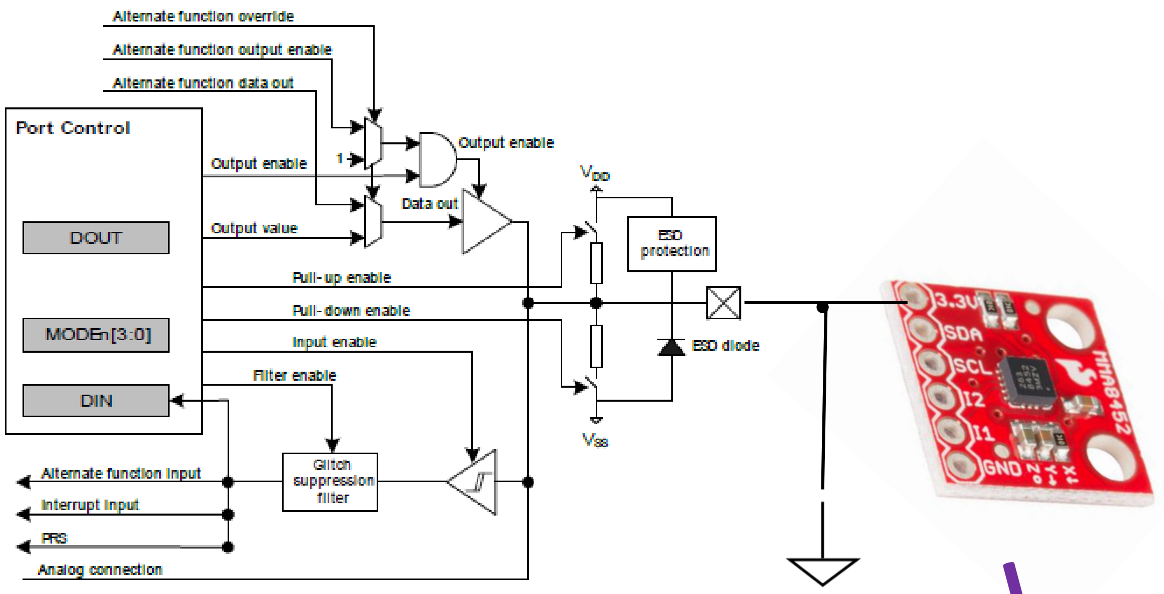
1. Make any changes required to the Managing Energy Mode Assignment

2. LETIMER0 should be set to the following conditions at startup / reset.
   a. Period = 3.0 seconds
   b. No LED heart beat requirement
   c. During the LETIMER0 period interrupt

Load Power Management via GPIO pin

- How long will it take the power line to stabilize
  - Using the recommended decoupling capacitance, 4.7uF
  - Calculate time to achieve VDD
    - $i = C\frac{dV}{dT}$
    - $dT = C\frac{dV}{i}$, 4.7uF $\frac{3.3V}{6mA}$
    - $dT = 2.59mS$
  - Verify that the power ramp meets the specifications of the external device

Figure 32.1. Pin Configuration



Power on ramp time	—		0.001		1000	ms

Verified

Load Power Management via GPIO pin

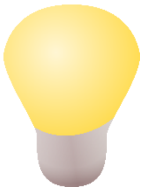
Enabling the external device pseudo code

- Turn power onto the external device
  - Set GPIO Power pin to 1
- Wait for power to stabilize + external boot time
  - For the MMA8452Q
    - 2.59mS + 500uS
    - 3.09mS

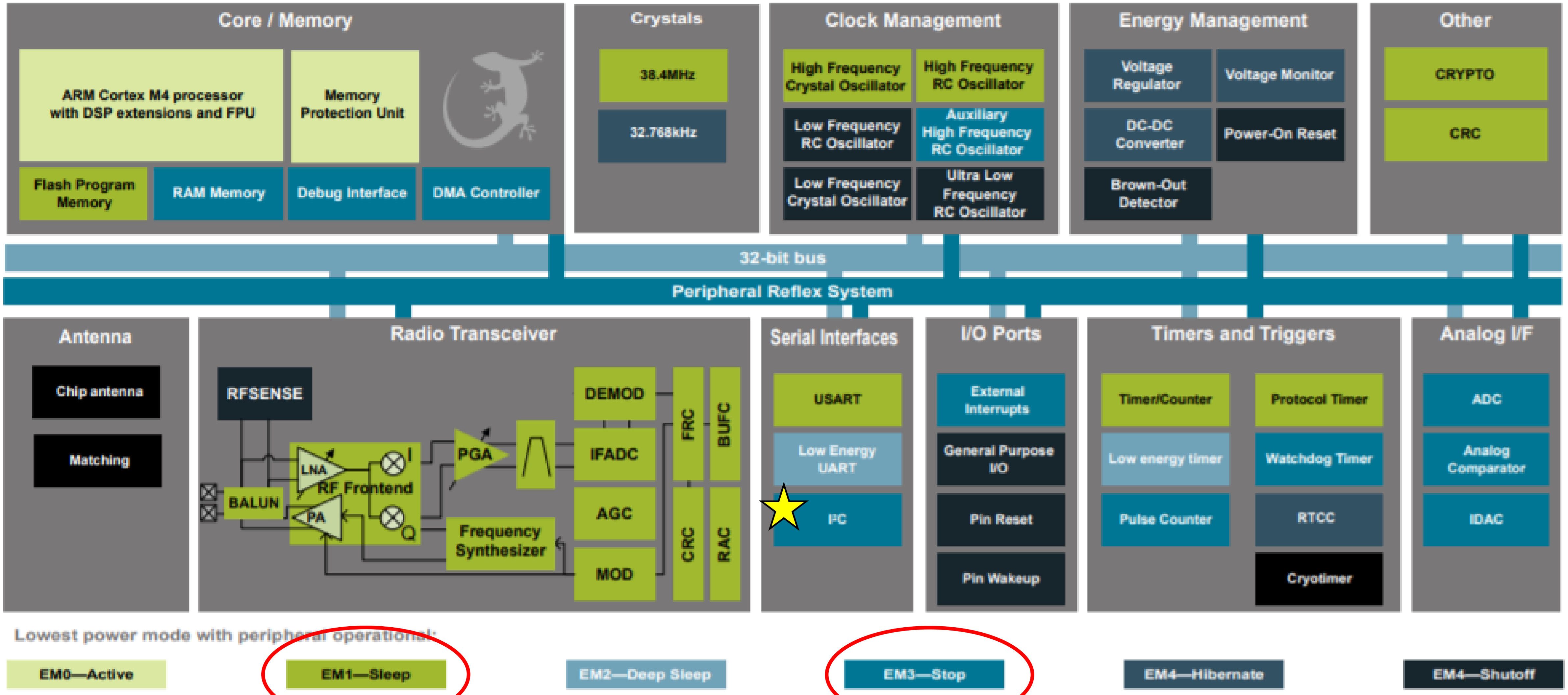
Boot time	Time from VDDIO on and VDD > VDD min until I^2C is ready for operation, Cbyp = 100 nF	Tbt	—	350	500	μs

- Enable GPIO I/O pins, such as SCL and SDA for I2C, on the MCU after peripheral to protect ESD diodes
- Initialize the device for operation
- Enable Interrupts if required
- Device is ready to be used!

Load Power Management via GPIO pin

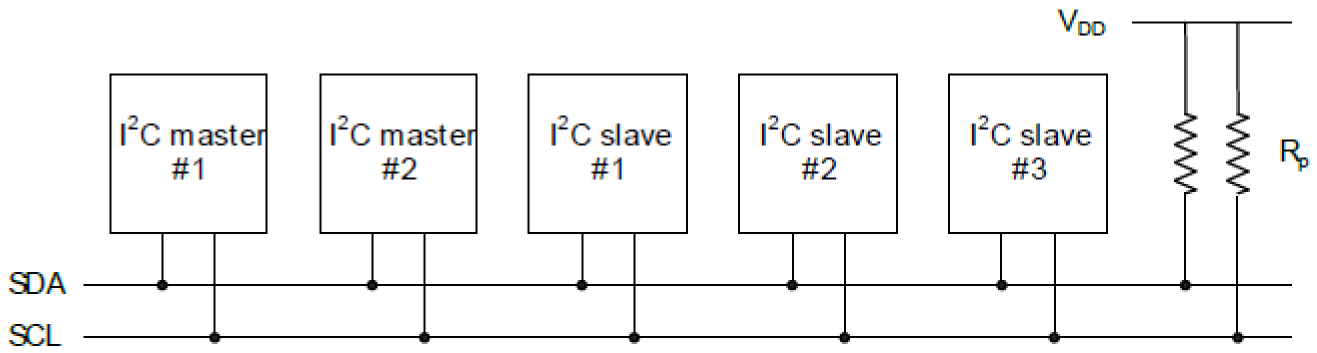


- Disabling the external device pseudo code
  - Disable Interrupts if used
  - Disable GPIO I/O pins, such as SCL and SDA for I2C, to protect ESD diodes
  - Turn off power to the GPIO Power pin by clearing the pin
    - Do not disable, but clear the pin to 0
  - Device is now deactivated and you are saving energy!


Core / Memory
ARM Cortex M4 processor with DSP extensions and FPU
Memory Protection Unit
Flash Program Memory
RAM Memory
Debug Interface
DMA Controller
Crystals
38.4MHz
32.768kHz
Clock Management
High Frequency Crystal Oscillator
High Frequency RC Oscillator
Low Frequency RC Oscillator
Auxiliary High Frequency RC Oscillator
Low Frequency Crystal Oscillator
Ultra Low Frequency RC Oscillator
Energy Management
Voltage Regulator
Voltage Monitor
DC-DC Converter
Power-On Reset
Brown-Out Detector
Other
CRYPTO
CRC
32-bit bus
Peripheral Reflex System
Antenna
Chip antenna
Matching
Radio Transceiver
RFSENSE
DEMOD
IFADC
AGC
MOD
FRC
BUFC
CRC
RAC
PGA
LNA
PA
BALUN
RF Frontend
I
Q
Frequency Synthesizer
Serial Interfaces
USART
Low Energy UART
I²C
I/O Ports
External Interrupts
General Purpose I/O
Pin Reset
Pin Wakeup
Timers and Triggers
Timer/Counter
Protocol Timer
Low energy timer
Watchdog Timer
Pulse Counter
RTCC
Cryotimer
Analog I/F
ADC
Analog Comparator
IDAC
Lowest power mode with peripheral operational:
EM0—Active
EM1—Sleep
EM2—Deep Sleep
EM3—Stop
EM4—Hibernate
EM4—Shutoff

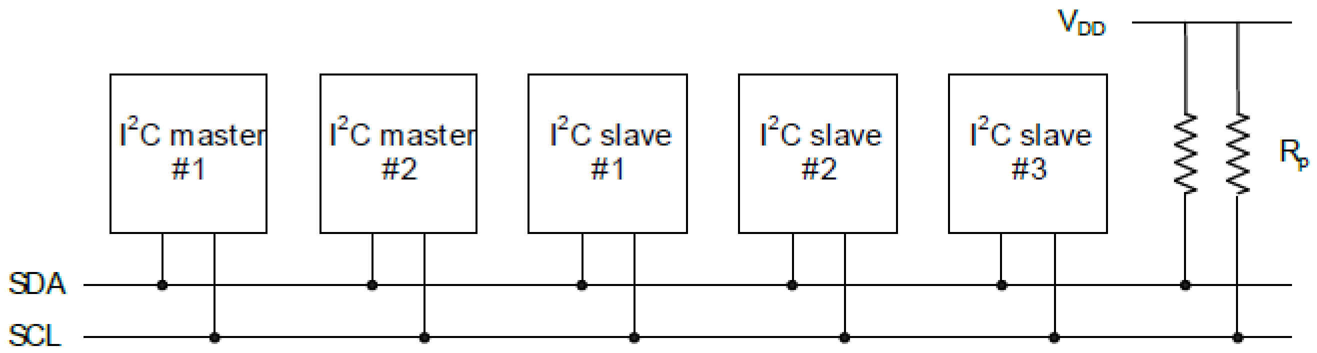
What is I2C?

- 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.

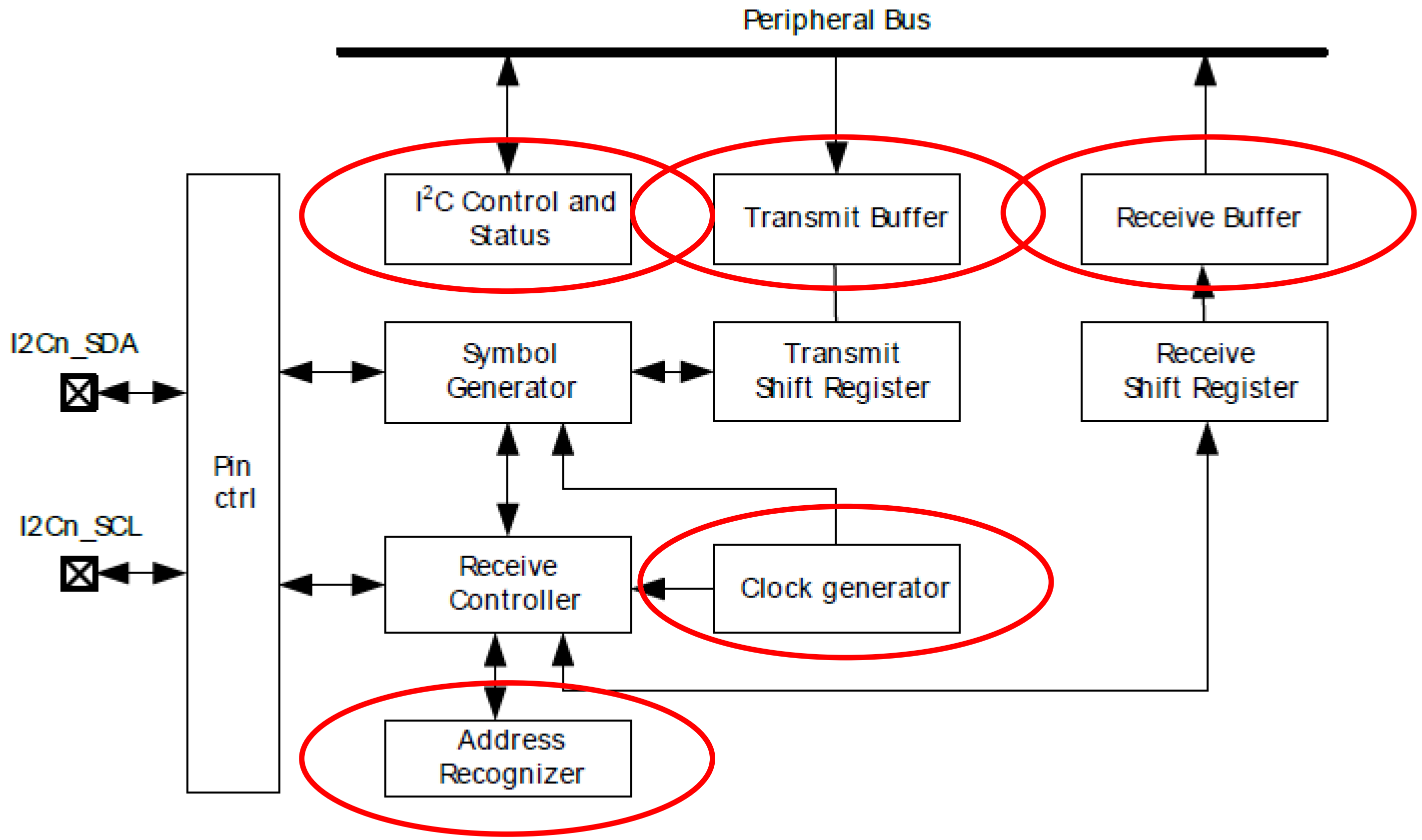


What is I2C?

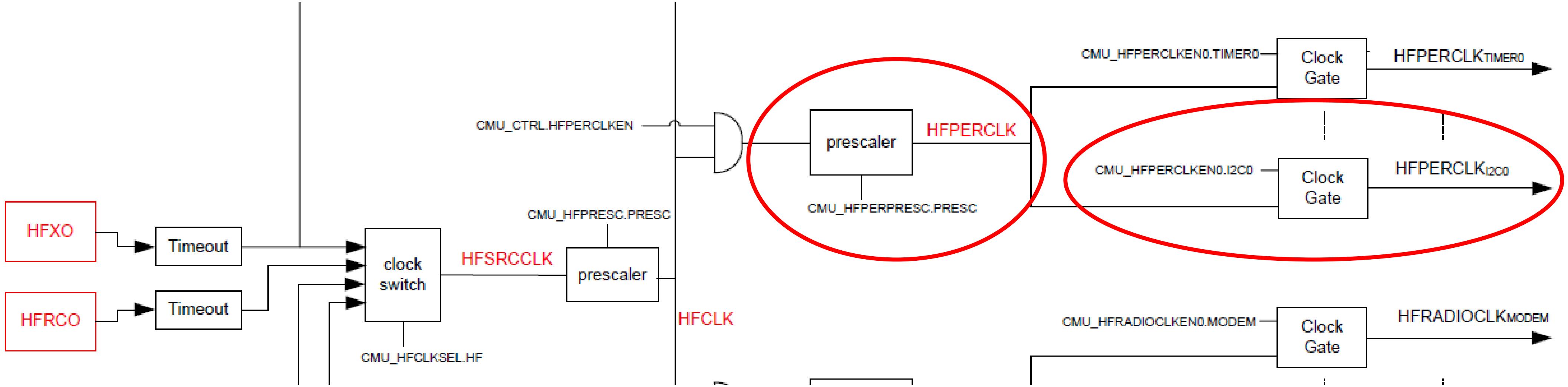
- Each device on the bus is addressable by a unique address
- The I2C master can address all the devices on the bus, including other masters
- Both the bus lines are open-drain. The maximum value of the pull-up resistor can be calculated as a function of the maximal rise-time tr for the given bus speed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



I2Cn peripheral block diagram



I2C



If the I2C clock input is the HFPERCLK, how can it work down into EM3?

Setting up the I2C

- What is the first thing we do to set up the I2C peripheral?
- 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

- After the clock tree has been established, what is the next step in setting up the I2C peripheral?
- Program the peripheral
  - 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

Table 6.1. CSP43 2.4 GHz Device Pinout

CSP Pin# and Name		Pin Alternate Functionality / Description				
Pin #	Pin Name	Analog	Timers	Communication	Radio	Other
A1	VREGSW	DCDC regulator switching node				
A2	VREGVDD	Voltage regulator VDD input				
A3	DECOUPLE	Decouple output for on-chip voltage regulator. An external decoupling capacitor is required at this pin.				
A4	IOVDD	Digital IO power supply.				
A6	PF0	BUSAX BUSBY	TIM0_CC0 #24 TIM0_CC1 #23 TIM0_CC2 #22 TIM0_CDTI0 #21 TIM0_CDTI1 #20 TIM0_CDTI2 #19 TIM1_CC0 #24 TIM1_CC1 #23 TIM1_CC2 #22 TIM1_CC3 #21 LE- TIM0_OUT0 #24 LETIM0_OUT1 #23 PCNT0_S0IN #24 PCNT0_S1IN #23	US0_TX #24 US0_RX #23 US0_CLK #22 US0_CS #21 US0_CTS #20 US0_RTS #19 US1_TX #24 US1_RX #23 US1_CLK #22 US1_CS #21 US1_CTS #20 US1_RTS #19 LEU0_TX #24 LEU0_RX #23 I2C0_SDA #24 I2C0_SCL #23	FRC_DCLK #24 FRC_DOUT #23 FRC_DFRAME #22 MODEM_DCLK #24 MODEM_DIN #23 MODEM_DOUT #22 MODEM_ANT0 #21 MODEM_ANT1 #20	PRS_CH0 #0 PRS_CH1 #7 PRS_CH2 #6 PRS_CH3 #5 ACMP0_O #24 ACMP1_O #24 DBG_SWCLKTCK #0
A7	PF1	BUSAY BUSBX	TIM0_CC0 #25 TIM0_CC1 #24 TIM0_CC2 #23 TIM0_CDTI0 #22 TIM0_CDTI1 #21 TIM0_CDTI2 #20 TIM1_CC0 #25 TIM1_CC1 #24 TIM1_CC2 #23 TIM1_CC3 #22 LE- TIM0_OUT0 #25	US0_TX #25 US0_RX #24 US0_CLK #23 US0_CS #22 US0_CTS #21 US0_RTS #20 US1_TX #25 US1_RX #24 US1_CLK #23 US1_CS #22 US1_CTS #21 US1_RTS #20	FRC_DCLK #25 FRC_DOUT #24 FRC_DFRAME #23 MODEM_DCLK #25 MODEM_DIN #24 MODEM_DOUT #23 MODEM_ANT0 #22 MODEM_ANT1 #21	PRS_CH0 #1 PRS_CH1 #0 PRS_CH2 #7 PRS_CH3 #6 ACMP0_O #25 ACMP1_O #25 DBG_SWDIOTMS #0

Helpful I2C hints from AN0011SW application note

```
/* Initializing I2Cn */
/* Output value must be set to 1 to not drive lines low... We set
*/
/* SCL first, to ensure it is high before changing SDA. */
GPIO_PinModeSet(I2Cn_SCL_Port, I2Cn_SCL_Pin, gpioModeWiredAnd, 1);
GPIO_PinModeSet(I2Cn_SDA_Port, I2Cn_SDA_Pin, gpioModeWiredAnd, 1);
```

Why are the pins set to WiredAnd and not PushPull?

Helpful I2C hints from AN0011SW application note

```
/* Initializing I2Cn */
/* Output value must be set to 1 to not drive lines low... We set */
/* SCL first, to ensure it is high before changing SDA. */
GPIO_PinModeSet(I2Cn_SCL_Port, I2Cn_SCL_Pin, gpioModeWiredAnd, 1);
GPIO_PinModeSet(I2Cn_SDA_Port, I2Cn_SDA_Pin, gpioModeWiredAnd, 1);

/* Toggle I2C SCL 9 times to reset any I2C slave that may require it */
for (int i=0;i<9;i++) {
   GPIO_PinOutClear(I2C1_SCL_Port, I2C1_SCL_Pin);
   GPIO_PinOutSet(I2C1_SCL_Port, I2C1_SCL_Pin);
   }
```

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

- With the I2C peripheral is set up, what is next?
- 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
    - Call BlockSleep mode right before accessing the I2C bus
    - The Blue Gecko can be an I2C Master in EM0 & EM1
    - The Blue 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

- With the interrupts set up, what is the last step in setting up the I2C peripheral?
- The I2Cn interrupt handler must be included
  - Routine name must match the vector table name:

```
Void I2Cn_IRQHandler(void) {
}
```

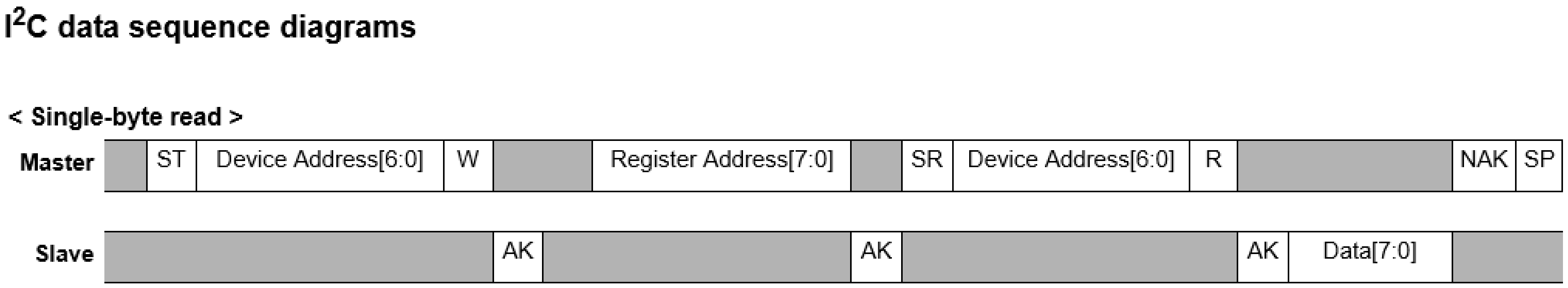
  - Inside this routine, you add the functionality that is desired for the I2Cn interrupts

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

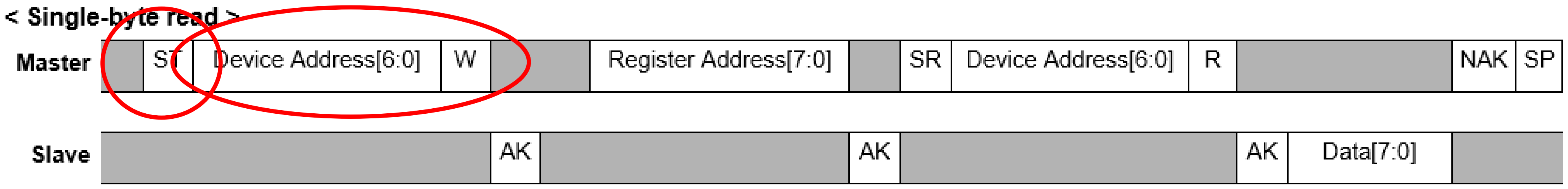


NXP MMA8452Q data sheet

Writing your own I2C driver

- Now, convert the visual diagram into a driver
- Prime the TX Buffer for the Start Command
  - I2Cn->TXDATA = (I2C_device_addr << 1) | R/W bit = 0 signifying write of address to the slave;
- Now send the Start Bit
  - I2Cn->CMD = I2Cn_CMD_START;

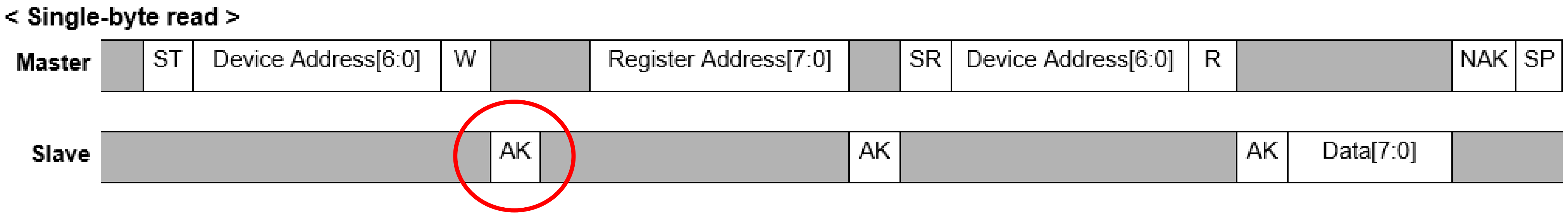
I²C data sequence diagrams



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;

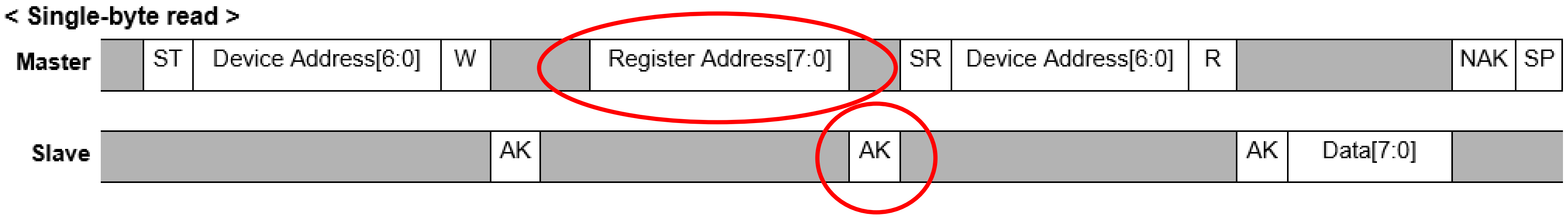
I^{2}C data sequence diagrams



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;

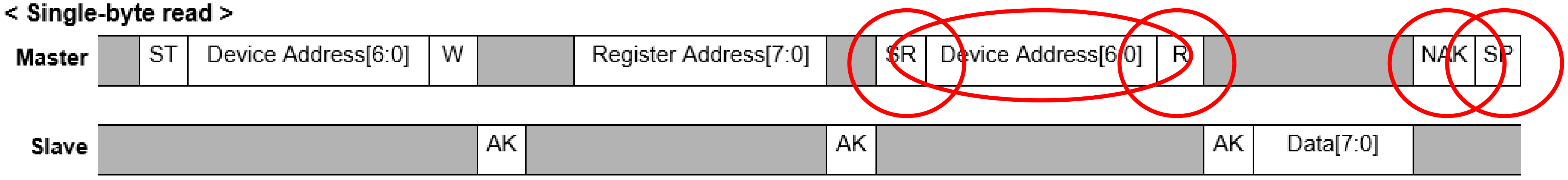
I²C data sequence diagrams



Writing your own I2C driver

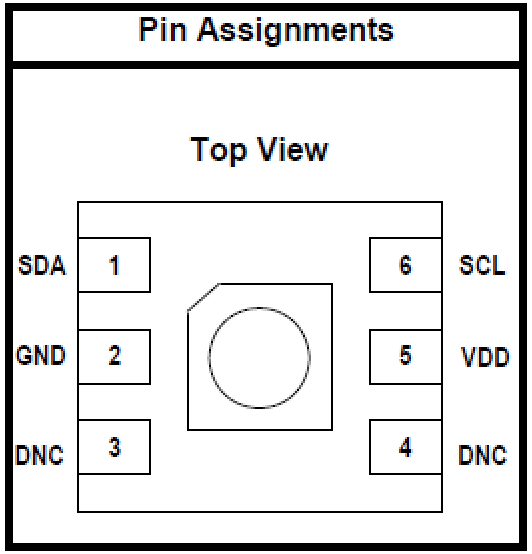
- Your driver continues down through out the visual diagram
  - Device Address
  - SR = Start Repeat
  - R = Read/Write bit set to 1 for Read Operation
  - NAK = NACK
  - SP = STOP

I^{2}C data sequence diagrams



Si7021 and Load Power Management

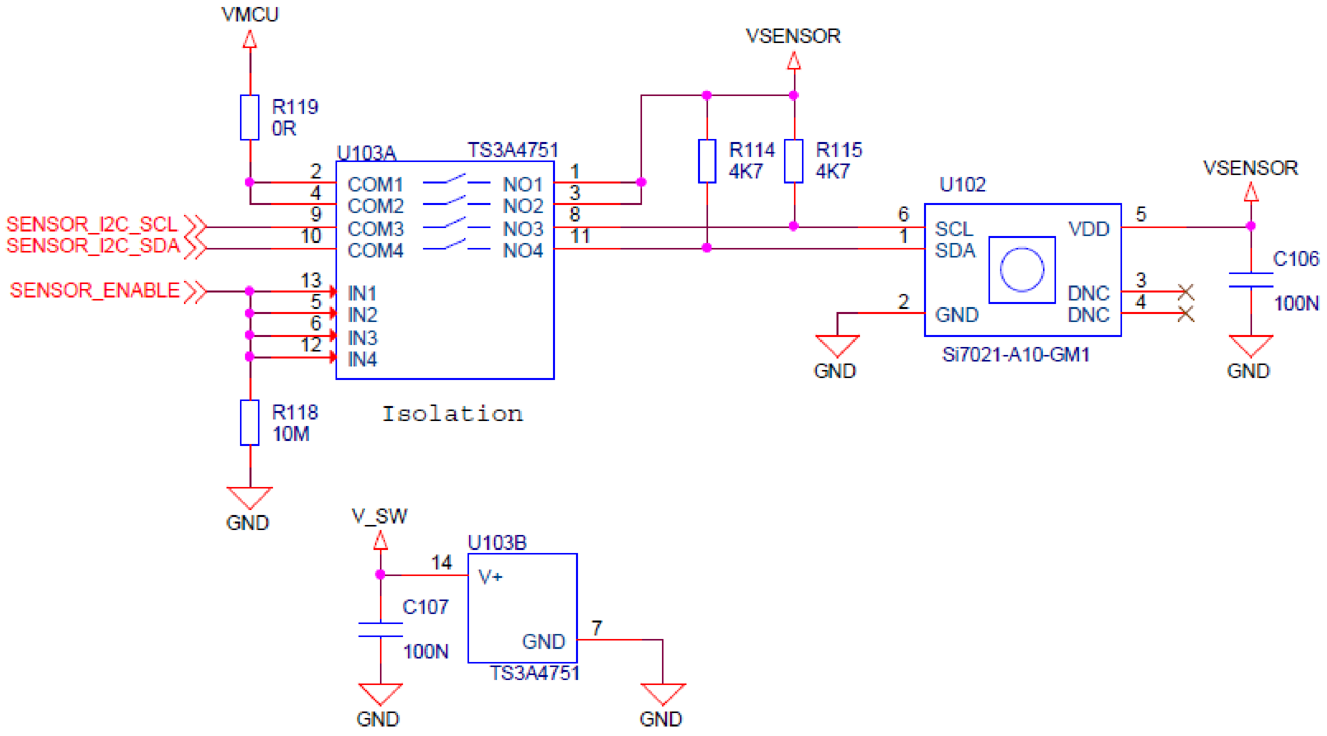
- What is missing from the package pin assignments?
- With no interrupt, how does the lack of an interrupt change the use of this device in a low energy environment?



Si7021 and Load Power Management

- What does this schematic tell us about Load Power Management?

Relative Humidity & Temperature Sensor



Scheduling

- This assignment and moving forward will require the use of a scheduler
- Main's while(1) loop will become:

```
while (1) {
	if (schedule_event == 0) sleep();
	if (schedule_event & event_1) {
			code servicing event 1;
			schedule_event &= ~event_1; }
	if (schedule_event & event_2) {
			code servicing event 2;
			schedule_event &= ~event_2;}
	…
}
```

Should you be handling these lines of code special?

Schedule_event is a global variable and to remove the possibility of a concurrency bug, accessing this global variable should be made atomic

Scheduling

- If we are servicing the interrupt in main's while(1) loop, what should the interrupt handler comprise of?
- Void Peripheral_IRQHander(void){

```
        int int_flag;
        disable_interrupts;
        int_flag = Peripheral->IF;
        Peripheral->IFC = int_flag
        If (int_flag & interrupt_1){
                Schedule_event |= peripheral_interrupt_1;}
        if (int_flag & interrupt_2){
                Schedule_event |= peripheral_interrupt_2;}
        enable_interrupts;
}
```

Scheduling

```
while (1) {
    if (schedule_event == 0) sleep();
    if (schedule_event & event_1) {
        code servicing event 1;
        schedule_event &= ~event_1; }
    if (schedule_event & event_2) {
        code servicing event 2;
        schedule_event &= ~event_2;}
    …
}
```

Why don't we use an elseif for the scheduler?

Si7021 and Load Power Management

- Load Power Management Turning ON sequence
  - Enable power to the Si7021 via the GPIO Sensor_Enable pin
  - Wait for the Si7021 to complete its Power On Rest, POR, and/or SCL and SDA pull ups to ramp up to “high” which ever is the longest period of time
  - Set the SCL and SDA gpio pins to “WiredAND”
  - Sequence SCL 9 times to reset all I2C peripherals on the bus
  - If the Blue Gecko I2C peripheral is busy, abort the operation to reset the I2C peripheral
  - Initialize the Si7021 to match the functionality required
  - Enable the function to request a Si7021 temperature measurement

Si7021 and Load Power Management

- Load Power Management Turning OFF sequence
  - Disable the application function to request a Si7021 temperature measurement
  - Take SCL and SDA off the I2C bus by placing the pins in “Disable” mode
    - Only good practice if there is no other I2C device on the bus. If there was another I2C, the application may still want to access this other I2C device
  - Set to “0” or clear the “Sensor_Enable” pin to turn off power to the I2C pullups and Si7021

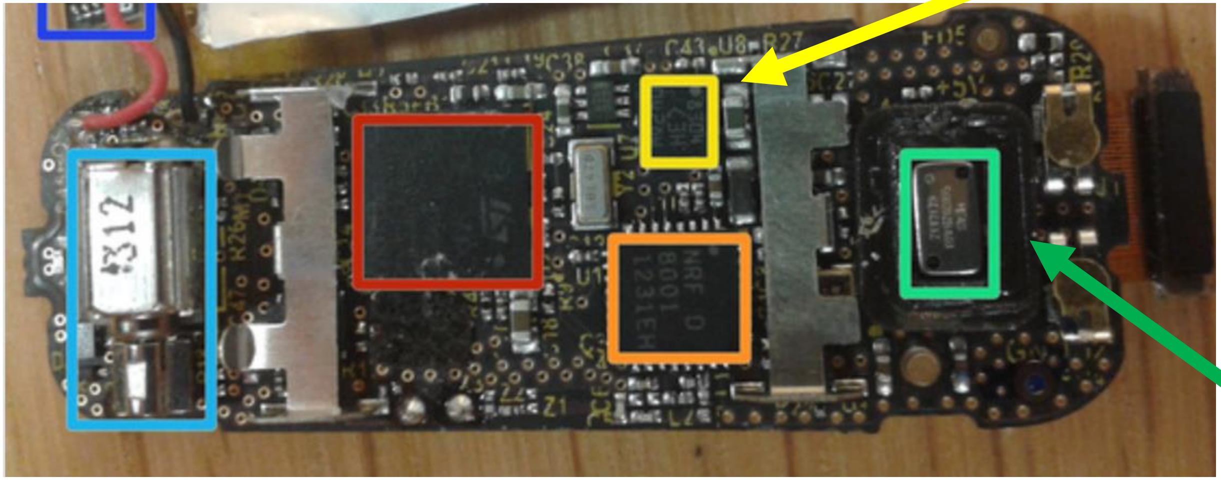
Sensors – For the low energy market

- Key Factors in a mobile sensor selection
    - End Product Features
        - What is the desired user experience?
        - What sensor features are required?
        - What is the energy budget for the product?
    - System considerations
        - What microcontroller interfaces available?
            - Analog IN
            - I2C
            - SPI
            - In what power mode?
        - What is the energy budget for the sensor?

Sensors – End Product Features

- What is the desired user experience?
  - What sensors are required to provide the user experience?
    - Example: Fitbit Odometer – Measures steps and staircases climbed
      - Sensors – Accelerometer and Altimeter

Accelerometer labelled 8304 AE D42 oW



Picture and p/ns from iFixIt

Measurement specialties MS5607-02BA03 altimeter

Sensors – End Product Features

- What sensor features are required?
  - To achieve the desired features, what are the required sensor specifications for the application?
  - Example: Accelerometer
    - Number of axis: 2, 3, etc.
    - Resolution: 10-bits, 12-bits, 14-bits, 16-bits, etc.
    - Range: +-2g, +-4g, +-8g, +-16g, etc.
    - Update rate: 1.25Hz, 5Hz, 10Hz, 20Hz, 40Hz, 400Hz, etc.

Freescale MMA8452Q, 3-Axis, 12-bit/8-bit Digital Accelerometer

Features

- 1.95V to 3.6V supply voltage
- 1.6V to 3.6V interface voltage
- ±2g/±4g/±8g dynamically selectable full-scale
- Output Data Rates (ODR) from 1.56 Hz to 800 Hz
- 99 µg/√Hz noise
- 12-bit and 8-bit digital output
- I^2C digital output interface
- Two programmable interrupt pins for six interrupt sources
- Three embedded channels of motion detection
  - Freefall or Motion Detection: 1 channel
  - Pulse Detection: 1 channel
  - Transient Detection: 1 channel
    - Orientation (Portrait/Landscape) detection with set hysteresis
    - Automatic ODR change for Auto-WAKE and return to SLEEP
    - High-Pass Filter Data available real-time
    - Self-Test
    - RoHS compliant
    - Current Consumption: 6 µA to 165 µA

Sensors – End Product Features

- What is the energy budget for the product?
  - Example: Fitbit Surge Watch
    - Battery life: last up to 7 days
    - GPS Battery life: last up to 10 hours
    - Battery type: Lithium-polymer
    - Charge time: One to two hours

Fitbit Surge photo & specifications from Fitbit Surge website

Sensors – System considerations

- What is the energy budget for the sensor?
  - Can drive the decision between active and passive sensor

Table 5. DC Characteristics

(Typical Operating Circuit, V_{DD} and V_{REG} = 1.8V, T_A = 25°C, unless otherwise noted.)

Parameter	Symbol	Conditions	Min	Typ	Max	Units
High Supply Voltage	V_{DD}		2.0	3.3	3.6	V
Low Supply Voltage	V_{REG}		1.71	1.8	2.75	V
Average Supply Current[(1)]	I_{DD}	Run Mode @ 1 ms sample period		393		µA
		Run Mode @ 2 ms sample period		199		µA
		Run Mode @ 4 ms sample period		102		µA
		Run Mode @ 8 ms sample period		54		µA
		Run Mode @ 16 ms sample period		29		µA
		Run Mode @ 32 ms sample period		17		µA
		Run Mode @ 64 ms sample period		11		µA
		Run Mode @ 128 ms sample period		8		µA
Measurement Supply Current	I_{DD}	Peak of measurement duty cycle		1		mA
Idle Supply Current	I_{DD}	Stop Mode		3		µA

Specification data from Freescale MPR121 capacitive touch sensor

Sensors – System considerations

- What microcontroller interfaces available?
  - Most common sensor interfaces
    - Analog In for passive sensors
      - LESENSE interface
    - Analog In for some active sensors such as audio and ambient light sensors
  - Active Sensors
    - I2C
    - SPI
    - UART

- **Communication interfaces**
  - 3× Universal Synchronous/Asynchronous Receiver/Transmitter
    - UART/SPI/SmartCard (ISO 7816)/IrDA/I2S
  - 2× Universal Asynchronous Receiver/Transmitter
  - 2× Low Energy UART
    - Autonomous operation with DMA in Deep Sleep Mode
  - 2× I^2C Interface with SMBus support
    - Address recognition in Stop Mode
  - Universal Serial Bus (USB) with Host & OTG support
    - Fully USB 2.0 compliant
    - On-chip PHY and embedded 5V to 3.3V regulator
- **Ultra low power precision analog peripherals**
  - 12-bit 1 Msamples/s Analog to Digital Converter
    - 8 single ended channels/4 differential channels
    - On-chip temperature sensor
  - 12-bit 500 ksamples/s Digital to Analog Converter
  - 2× Analog Comparator
    - Capacitive sensing with up to 16 inputs
  - 3× Operational Amplifier
    - 6.1 MHz GBW, Rail-to-rail, Programmable Gain
  - Supply Voltage Comparator
- **Low Energy Sensor Interface (LESENSE)**
  - Autonomous sensor monitoring in Deep Sleep Mode
  - Wide range of sensors supported, including LC sensors and capacitive buttons

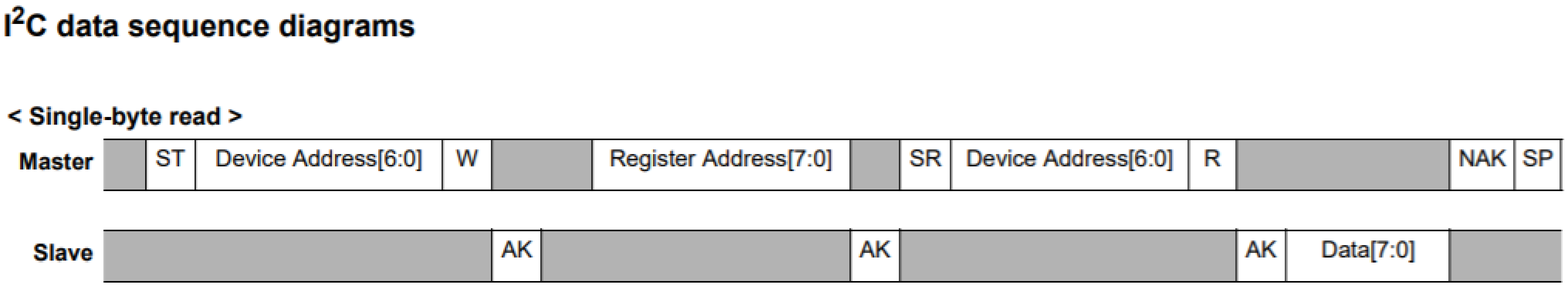
Communication List from Silicon Labs' EFM32LG data sheet

What Digital Serial Bus is the lowest energy?

- The most common sensor buses are:
  - I2C
  - SPI
  - UART
- Energy = Power x Time

Comparing the Time of a single byte transfer

- I2C



- A total of 40 cycles required for a single read transfer
- Time required:
  - 400KHz I2C = 100uS
  - 1,000KHz I2C = 40uS

Comparing the Time of a single byte transfer

- SPI
- 16 total cycles require for a single read transfer
- Time required:
  - 10,000 KHz = 1.6uS

The basic read operation waveform for 4-wire configuration is depicted in figure 15:

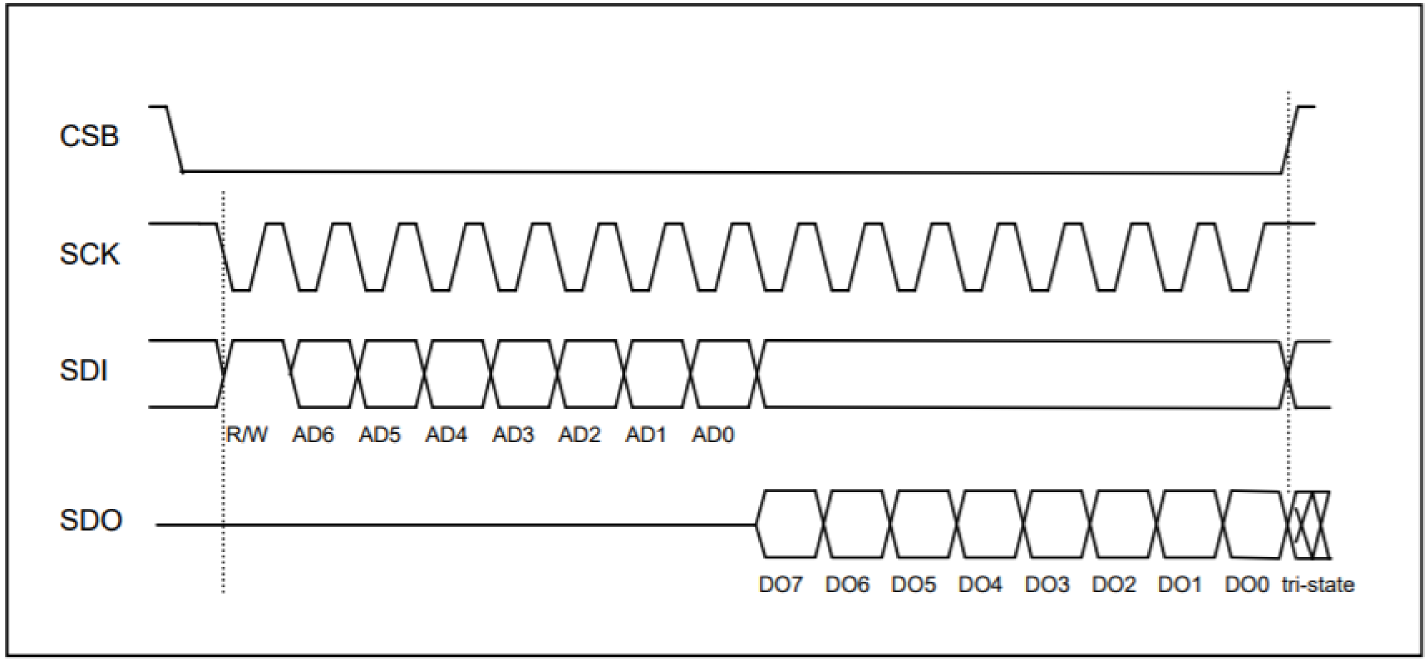


Figure 15: 4-wire basic SPI read sequence (mode ´11´)

Comparing the Time of a single byte transfer

- UART

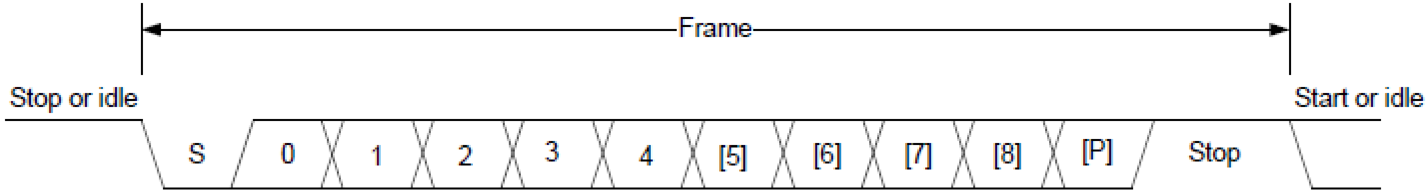


Figure 18.2. USART Asynchronous Frame Format

- A total of 11 cycles required for a single read transfer
- Time required:
  - 32.768KHz UART = 336uS
  - 115.2KHz UART = 95.5uS

Comparing the Power of a single byte transfer

- We will use the Silicon Labs Blue Gecko as a reference micro-controller due to its low energy design
- I2C can run as low as EM1 with DMA

Current consumption in EM1 Sleep mode with all peripherals disabled	I_{EM1}	38.4 MHz crystal[1]	—	65	—	µA/MHz
		38 MHz HFRCO	—	35	38	µA/MHz
		26 MHz HFRCO	—	37	41	µA/MHz
		1 MHz HFRCO	—	157	275	µA/MHz

- Standard HFXO crystal = 38.4MHz
  - Power = 65uA * 38.4 = 2.5mA
- Assume 50% of the time the pull ups are active (2 pull-ups: SCL and SDA)
  - 2* 0.50 * (3.3v / 10Kohm) = 0.330mA
- Total current:
  - EM1 + Pull ups = 2.83mA
- Total Power:
  - V * I = 3.3 * 2.83mA = 9.34mW

Comparing the Power of a single byte transfer

- We will use the Silicon Labs Blue Gecko as a reference micro-controller due to its low energy design
- SPI can run as low as EM1 with DMA

Current consumption in EM1 Sleep mode with all peripherals disabled	I_{EM1}	38.4 MHz crystal[1]	—	65	—	µA/MHz
		38 MHz HFRCO	—	35	38	µA/MHz
		26 MHz HFRCO	—	37	41	µA/MHz
		1 MHz HFRCO	—	157	275	µA/MHz

- Standard HFXO crystal = 38.4MHz
  - Power = 65uA * 38.4 = 2.5mA
- Total current
  - EM1 = 2.5mA
- Total power
  - V * I = 3.3 * 2.5mA =8.25mW

Comparing the Power of a single byte transfer

- We will use the Silicon Labs Blue Gecko as a reference micro-controller due to its low energy design
- LEUART can run as low as EM2 with DMA

Current consumption in EM2 Deep Sleep mode.	I_{EM2}	Full RAM retention and RTCC running from LFXO	—	3.3	—	μA
		4 kB RAM retention and RTCC running from LFRCO	—	3	6.3	μA

- HFXO oscillator has been turned off
- Total current
  - EM1 = 2.5mA
  - EM2 = 6.3uA
- Total Power
  - EM1 = 3.3v * 2.5mA = 8.25mW
  - EM2 = 3.3v * 0.0063mA = 0.02mW

What is the lowest Energy digital serial interface?

Digital Serial Interface	Power	Time	Energy
I2C @ 400KHz	9.34mW	100uS	934nJ
I2C @ 1,000KHz	9.34mW	40uS	374J
SPI @ 10,000KHz	8.25mW	1.6uS	13nJ
UART @ 115.2KHz	8.25mW	95.5uS	788nJ
LEUART @ 32.768KHz	0.02mW	336uS	7nJ

Other Digital Serial Bus Considerations

- I2C
  - Advantage: Addressable address bus up to 128 devices
    - Supports multiple sensors without increasing GPIO pin utilization or additional MCU resources
- SPI
  - Disadvantage: Requires additional GPIO pin for Chip Select for each additional addressable device
- UART
  - Disadvantage: Requires additional UART resource and GPIO pins for each device