ECEN 5023-001, -001B, -740

Mobile Computing & IoT Security

Lecture #8

9 February 2017

Black board

Agenda

- Class Announcements
- ACMP and ULFRCO self calibration assignment review
- Reducing ADC energy with DMA
- DMA
- DMA – arbitration
- Revisiting ADC output to degrees C routine
- Sensors for the low power mobile market
- Load Power Management
  - Introduction to I/O pin ESD diodes

Class Announcements

- Quiz #4 is due at 11:59pm on Sunday, February 12th, 2017
- Reducing ADC energy with DMA is due on Wednesday, February 12th, 2017
- The following slack channels have been created:
  - Assignment3
  - adc
  - dma

Self-calibrating UFRCO assignment

Question portion of assignment (total of 5 pts)

- Question 1: As long as it is outside 2.45 and 2.55 seconds, (0.5pts)
- Question 2: With calibration, it should be 2.42 to 2.58 seconds (+/- 3%) (1.0pt)
- Question 3: < +/-3% or less than 0.075 seconds (0.5pts)
- Question 4: 6.5, 6.6, or 6.7 (0.4 pts), average current <= 1.0uA (0.6pts) (total 1.0pt)
- Question 5: 3.4, 3.5, or 3.6 (0.4 pts), average current <= 500uA (0.6pts) (total 1.0pt)
- Question 6: Yes, the LED turns on (0.25pts)
- Question 7: Yes, the LED turns off (0.25pts)

Self-calibrating UFRCO assignment

Functional Code review portion of the assignment

- 1 pt total for the following defined statements:
  - 0.5 points if code used #defined statements for the following:
    - **#define** Light_Excite_port gpioPortD
    - **#define** Light_Excite_pin 6U
    - **#define** Light_Sensor_port gpioPortC
    - **#define** Light_Sensor_pin 6U
  - 0.5 points if code used #defined statements for the following or similar (they could be doing the shifting in the program):
    - **#define** Light_Sensor_ACMP_Channel acmpChannel6
    - **#define** Light_Sensor_ACMP_Ref acmpChannelVDD
    - **#define** Light_Sensor_Darkness_Ref (2U << 8)
    - **#define** Light_Sensor_Light_Ref (61U << 8)

Self-calibrating UFRCO assignment

Functional Code review portion of the assignment

- 0.5 point if there is a comment disclaiming the origins of the sleep routines and specifies that the particular sleep routines are covered by this IP statement/comment.
- 0.5 point if the code runs as specified in EM2
  - Measure period should be 2.42 to 2.58 seconds
- 1 point if the code runs as specified in EM3
  - Measure period should be 2.42 to 2.58 seconds
- 0.5 points if the measured time from LED excite to measurement is 4.0mS+ in EM3
  - Due to integer math, if they are not careful, the excite time could be less than the required 4mS
- 1 pt if while running and the LED is OFF, the Energy Score is greater than 6.5 while running on your board in EM3
- 0.5 points if TIMER0 and TIMER1 clocks are turned off after calibration

Reducing ADC energy with DMA Assignment #3

Objective: Develop the code for the Leopard Gecko to indicate whether the on board temperature indicates a temperature below a low limit set point or indicates a temperature above another set point.

- The temperature used to determine whether the temperature has gone outside its temperature settings, will be an average of 750 temperature readings.
- To save energy by not requiring the CPU to be on during these 750 readings, the results from the ADC should be sent to memory via DMA and only after 750 readings should the CPU wake up, calculate the average, and turn on LED1 if outside of the temperature limit settings.

Reducing ADC energy with DMA Assignment #3

Instructions:

1. Make any corrections to assignment #1 that are necessary
2. Make any corrections to assignment #2 that are necessary
3. Read the Leopard Gecko Reference Manual sections:
   a. ADC
   b. DMA
4. Review the Silicon Labs' EM32 application notes for:
   a. ADC
   b. DMA
5. The LETIMER0 will be set to energy mode EM3 with a period of 5.5 seconds
6. This assignment is being built upon the ACMP and Ambient Light Sensor assignment, so assignment #2 should still be working in assignment #3 with the change to the period to 5.5 seconds.

Reducing ADC energy with DMA Assignment #3

Instructions:

7. Program the ADC0 peripheral to perform the following ADC conversations:
    a. ADC input channel the Leopard Gecko's internal Temperature Sensor
    b. Conversions should be taken at a rate around 20,000 conversions / second
    c. 12-bit resolution
    d. The ADC should be set up to utilize DMA to store the 750 conversion results
    e. BlockSleepMode(ADC_EM) should be called every time that the 750 conversions are initiated
    f. UnBlockSleepMode(ADC_EM) should be called every time that the 750 conversions are completed to enable the system to go back to EM3 energy mode
    g. The 750 conversions should be initiated via the ADC_Start(ADC0, adcStartSingle);

8. The DMA peripheral will need to be programmed as well as the DMA channel for the ADC should be configured
    a. Set ADC DMA arbitration R bit to 0
    b. ADC DMA request should use the ADC Single DMA request line

Reducing ADC energy with DMA Assignment #3

Instructions:

9. Program the routine to convert the ADC temperature sensor inputs into a temperature in degrees C
    a. Please credit the IP for this routine to Silicon Labs

10. Every 5.5 seconds the LETIMER should initiate the 750 ADC conversions
    a. To initiate these conversions, the DMA for these memory transfers will need to be set up using the DMA_Activate_Basic command
    b. After the DMA has been set up, the BlockSleep(ADC_EM) should be called before the ADC_Start() command
    c. In the DMA call back routine, the ADC should be turned off, and then the UnBlockSleep(ADC_EM) routine
    d. Upon completing the ADC conversions and memory transfers, average the 750 conversions and translate the result into a temperature
    e. If the temperature is outside the temperature limits, LED1 should be turned on
    f. If the temperature then goes inside the limits, the LED1 should be turned off

Reducing ADC energy with DMA Assignment #3

Instructions:

11. There should be a #define where the DMA should be turned off and the conversions are done via a poll in LETIMER0 instead of DMA.
    a. Similar to the DMA, after the 750 conversions, the ADC should be disabled and UnBlockSleep(ADC_EM) executed
    b. Same as the DMA, in this mode, the temperature should be calculated and if outside the temperature bounds, the LED1 should be turned on
    c. Same as in the DMA, in this mode, when the temperature transactions back within the temperature bounds, the LED1 should be turned off

12. #defined statements
    a. Lower Temperature Limit set to 15
    b. Upper Temperature Limit set to 35
    c. All appropriate ADC peripheral settings
    d. All appropriate DMA peripheral settings
    e. All appropriate ADC DMA channel settings

Reducing ADC energy with DMA Assignment #3

Questions:

In a separate document to be placed in the drop box with the program code, please answer the following questions:

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

1. 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 EM2 and not using DMA and what is the average current when no LED is on and no measurements are being taken? Also, what is the average current for one 5.5 second period?

2. 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 EM2 and <u>using</u> DMA and what is the average current when no LED is on and no measurements are being taken? Also, what is the average current for one 5.5 second period?

3. Measure the time of the ADC doing its DMA. How many milliseconds is the ADC DMA operation?

Reducing ADC energy with DMA Assignment #3

Questions:

4. 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 not using DMA and what is the average current when no LED is on and no measurements are being taken? Also, what is the average current for one 5.5 second period?

5. 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 DMA and what is the average current when no LED is on and no measurements are being taken? Also, what is the average current for one 5.5 second period?

6. Does LED0 latched on when darkness is detected in EM3 and using DMA for ADC transfers?

7. Does LED0 turn latch off when light is detected in EM3 and using DMA for ADC transfers?

Reducing ADC energy with DMA Assignment #3

8. Does LED1 turn on when below temperature limit in EM3 and using DMA for ADC transfers?
   a. You can set the temperature limits artificially high to stimulate this condition

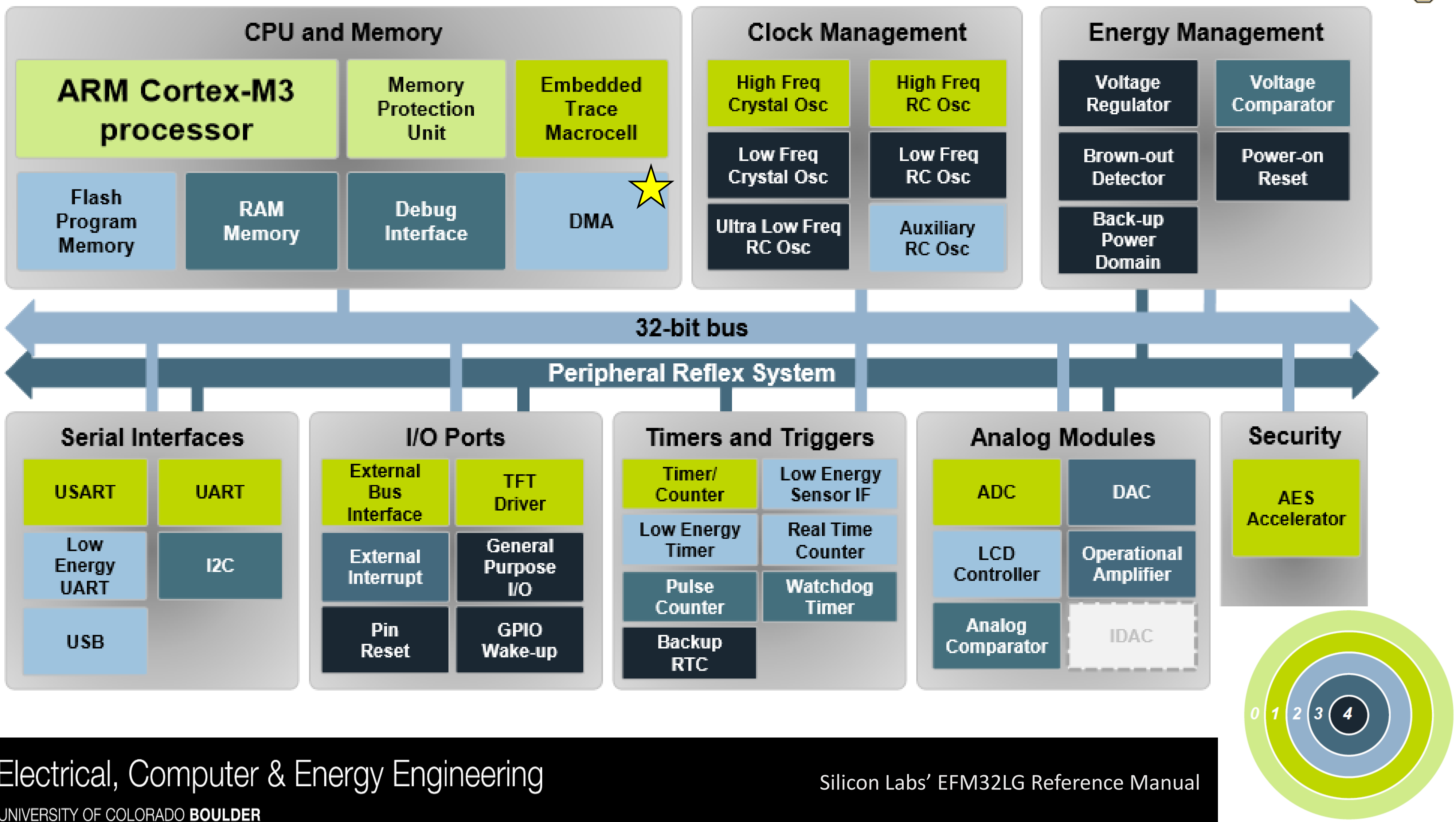
9. Does LED1 turn on when above the temperature limit in EM3 and using DMA for ADC transfers?
   a. You can set the upper temperature limits artificially low to stimulate this condition

Deliverables:

1. One document that provides the answers to Assignment #3

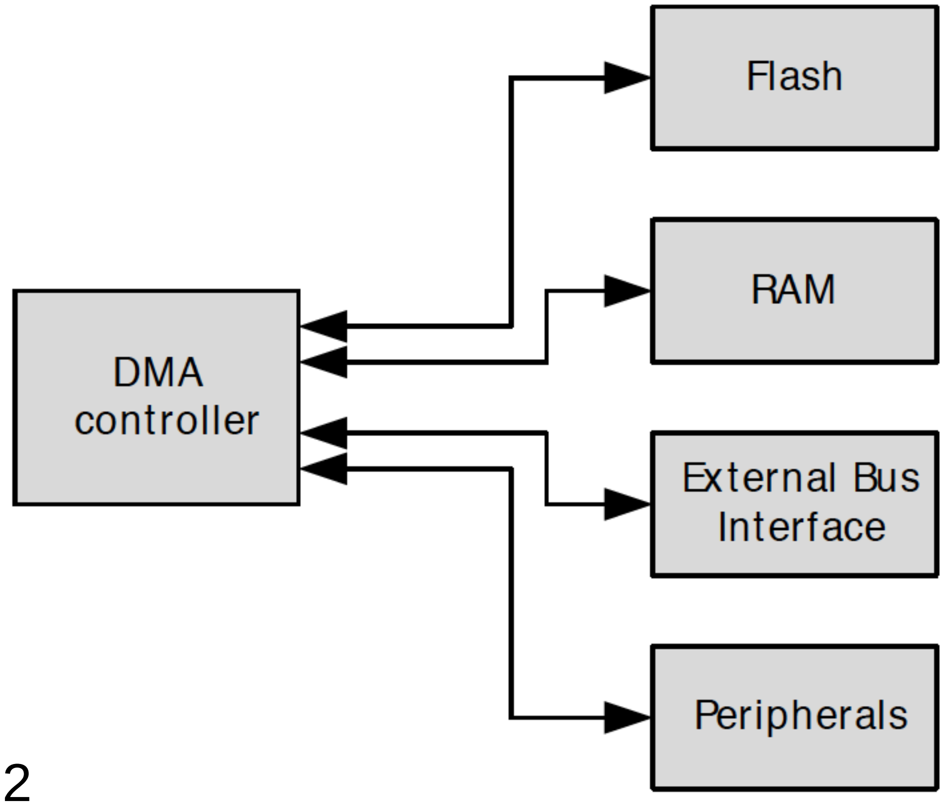
2. Another document that contains your .c project source and header files with LETIMER0 set to EM3 mode

BONUS: If you can beat the professors Energy Score on question 4 running on the same STK3600 dev kit, you will get 1 bonus points. The assignment must also be turned in on time.


CPU and Memory
ARM Cortex-M3 processor
Memory Protection Unit
Embedded Trace Macrocell
Flash Program Memory
RAM Memory
Debug Interface
DMA
Clock Management
High Freq Crystal Osc
High Freq RC Osc
Low Freq Crystal Osc
Low Freq RC Osc
Ultra Low Freq RC Osc
Auxiliary RC Osc
Energy Management
Voltage Regulator
Voltage Comparator
Brown-out Detector
Power-on Reset
Back-up Power Domain
32-bit bus
Peripheral Reflex System
Serial Interfaces
USART
UART
Low Energy UART
I2C
USB
I/O Ports
External Bus Interface
TFT Driver
External Interrupt
General Purpose I/O
Pin Reset
GPIO Wake-up
Timers and Triggers
Timer/ Counter
Low Energy Sensor IF
Low Energy Timer
Real Time Counter
Pulse Counter
Watchdog Timer
Backup RTC
Analog Modules
ADC
DAC
LCD Controller
Operational Amplifier
Analog Comparator
IDAC
Security
AES Accelerator
0
1
2
3
4

DMA – Direct Memory Access

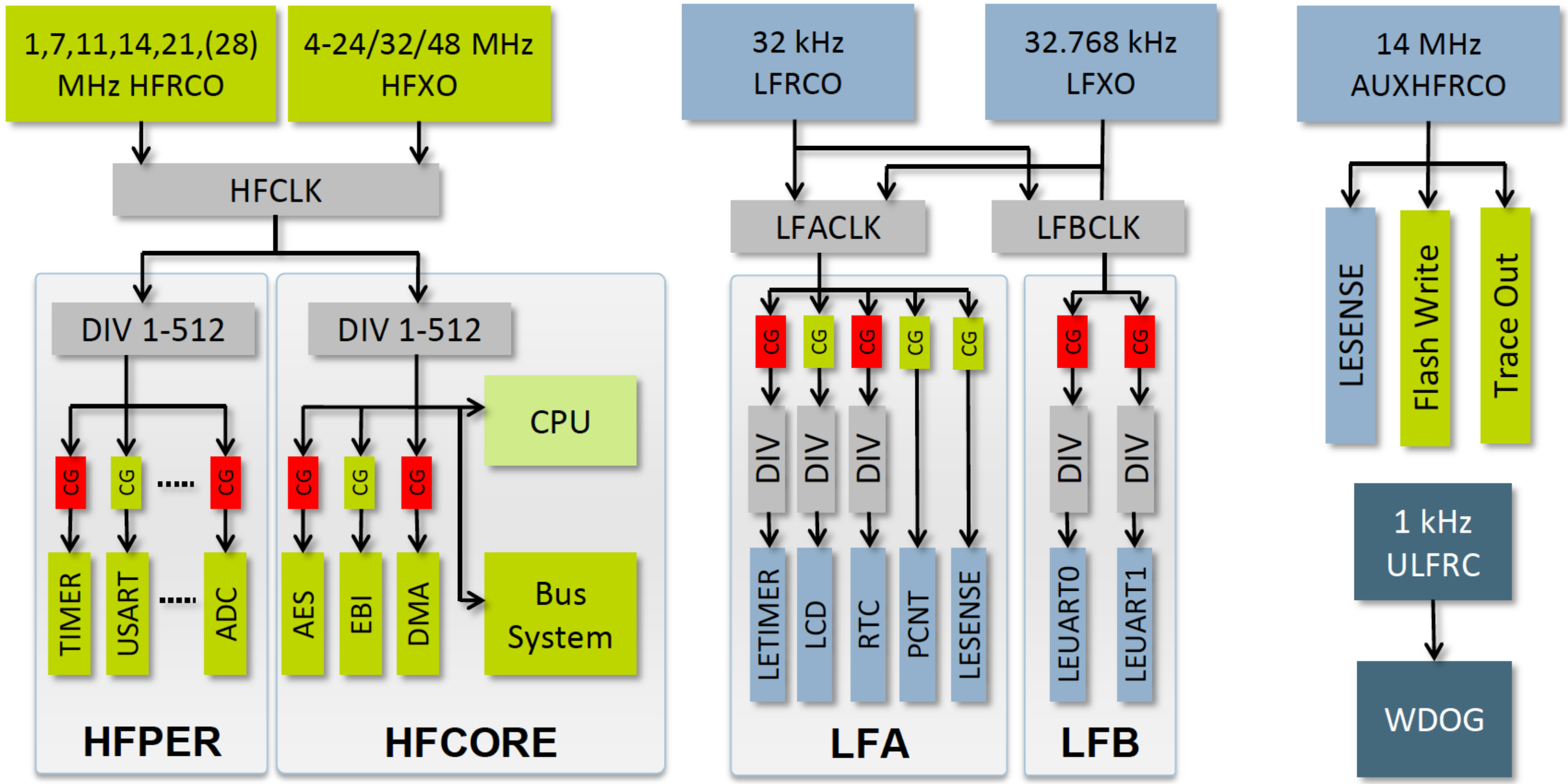
- The DMA controller is accessible as a memory mapped peripheral
- Possible data transfers include
  - RAM/EBI/Flash to peripheral
  - RAM/EBI to Flash
  - Peripheral to RAM/EBI
  - RAM/EBI/Flash to RAM/EBI
- A DMA channel can be triggered by any of several sources:
  - Communication modules (USART, UART, LEUART)
  - Timers (TIMER)
  - Analog modules (DAC, ACMP, ADC)
  - External Bus Interface (EBI)
  - Software
- DMA transfers to/from LEUART are supported in EM2



Clocks and Oscillators

DMA is connected to the HFPERCLK, which is always running in EM0 & EM1, so what does this mean?

You will need to enable the HFPER clock tree



But, you still need to enable the clock to the DMA!

DMA – Functional description

- DMA transfers data between peripherals and memory
- DMA benefits:
  - Increases system performance by off-loading the processor from copying large amounts of data
  - Reduces frequent interrupts to be serviced
  - Reduces system energy consumption
    - by performing transfers in EM1 and EM2 while the core is asleep
    - And reducing the number of interrupt service handling calls
- DMA descriptors define what the DMA peripheral will perform when one of its channels triggers a DMA request
  - These descriptors reside in system memory and must be configured before a DMA transfer is requested
- DMA engine supports several different modes; basic, ping-pong, loop, and 2-D

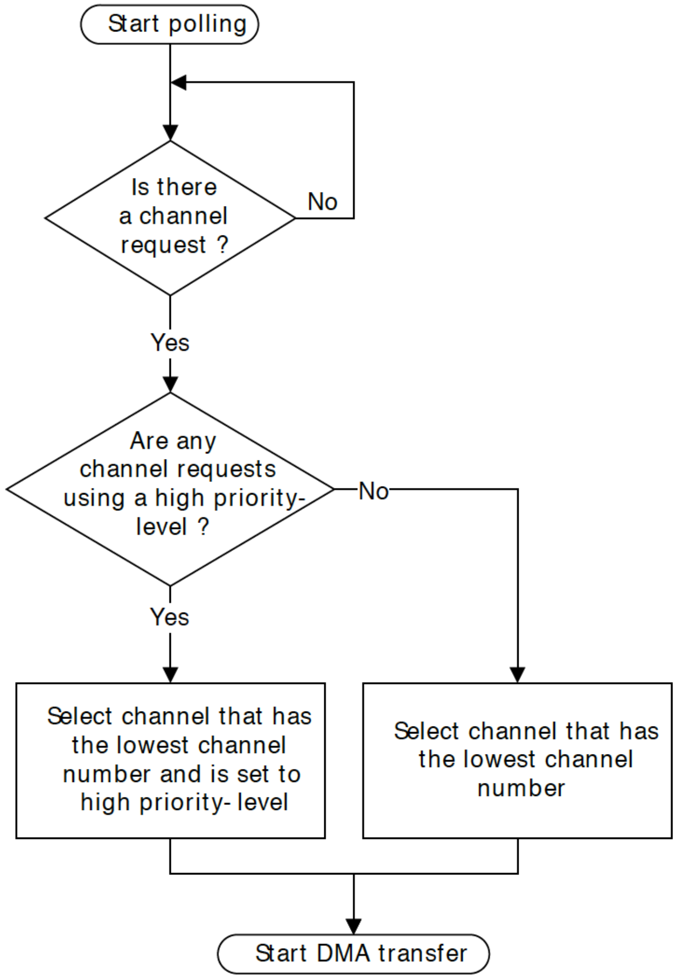
DMA - Arbitrate

- Arbitrate is requesting the number of DMA transfers that the peripheral will perform upon receiving the DMA bus
- Four bits per DMA channel found in the channel control structure defines how many DMA transfers will occur before it releases the DMA bus to be granted to another peripheral
- This mechanism is used to optimize DMA for peripherals that are requesting multiple memory transfers and not being required to renegotiate the DMA bus
  - Low latency examples: Communication peripherals such as USB, UARTs, DAC
  - Latency may not be an issue: ADC, ACMP

DMA - Priority

- During each DMA bus negotiations cycle, the DMA engine decides which channel is awarded the DMA bus based on the channel's prioritization
- Each channel's priority level can be set in the DMA_CHPRIS register, HIGH or DEFAULT
- A peripheral's DMA priority is based on the channel number specified to the peripheral and the channel priority assigned to the channel in the DMA_CHPRIS register
- The DMA channel that is awarded the DMA bus goes to the channel with the following:
  - HIGH priority channel with the lowest channel number
  - If no HIGH priority, DEFAULT priority channel with lowest channel number

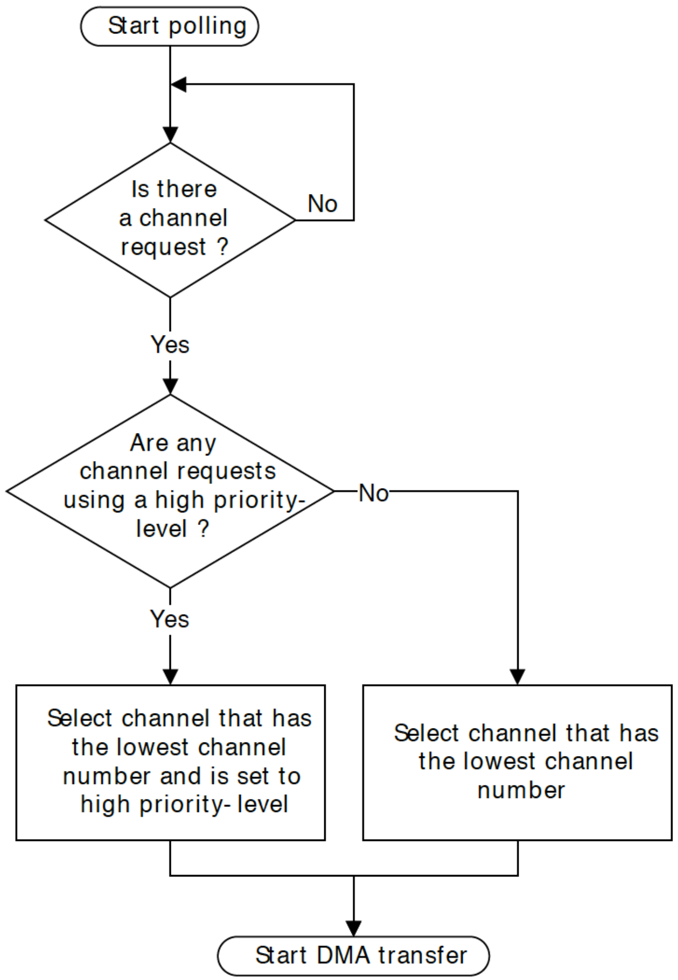
Figure 8.2. Polling flowchart



DMA - Priority

- During each DMA bus negotiations cycle, the DMA engine decides which channel is awarded the DMA bus based on the channel's prioritization
- Each channel's priority level can be set in the DMA_CHPRIS register, HIGH or DEFAULT
- A peripheral's DMA priority is based on the channel number specified to the peripheral and the channel priority assigned to the channel in the DMA_CHPRIS register
- The DMA channel that is awarded the DMA bus goes to the channel with the following:
  - HIGH priority channel with the lowest channel number
  - If no HIGH priority, DEFAULT priority channel with lowest channel number

Figure 8.2. Polling flowchart



DMA - Priority

Note: There is a channel 0 high priority, but was on the previous page which is the highest priority DMA peripheral

- A DMA channel is given to a single peripheral
  - So there will never be a channel that can be set to high and default priority
- The lower the channel number, the higher priority with the same priority settings
- Which peripheral would be awarded the DMA bus if both are requesting the bus simultaneously?
  - Channel 3-high priority   Channel 10-high priority
- Which peripheral would be awarded the DMA bus if both are requesting the bus simultaneously?
  - Channel default priority   Channel 10-high priority

Channel number	Priority level setting	Descending order of channel priority
1	High	-
2	High	-
3	High	-
4	High	-
5	High	-
6	High	-
7	High	-
8	High	-
9	High	-
10	High	-
11	High	-
0	Default	-
1	Default	-
2	Default	-
3	Default	-
4	Default	-
5	Default	-
6	Default	-
7	Default	-
8	Default	-
9	Default	-
10	Default	-
11	Default	Lowest-priority DMA channel

DMA – Arbitration example

- LEUART is set with arbitration of b0000, 1, high priority DMA channel 0
- USB is set with arbitration of b0010, 4, high priority DMA channel 5
- Assume that the USB is transferring a large block of data, it could prevent the LEUART from accessing the DMA resources if it was set to arbitrate after 1024 DMA transfers by not giving the DMA bus up to arbitration while the LEUART required access to memory which resulted in the LEUART being starved of data
- With the USB forced to arbitrate after every 4 DMA transfers insures that the LEUART will have access to the DMA bus due to its higher priority

Table 8.1. AHB bus transfer arbitration interval

R_power	Arbitrate after x DMA transfers
b0000	x = 1
b0001	x = 2
b0010	x = 4
b0011	x = 8
b0100	x = 16
b0101	x = 32
b0110	x = 64
b0111	x = 128
b1000	x = 256
b1001	x = 512
b1010 - b1111	x = 1024

Black board

DMA – Cycle types

Table 8.3. DMA cycle types

cycle_ctrl	Description
b000	Channel control data structure is invalid
b001	Basic DMA transfer
b010	Auto-request
b011	Ping-pong
b100	Memory scatter-gather using the primary data structure
b101	Memory scatter-gather using the alternate data structure
b110	Peripheral scatter-gather using the primary data structure
b111	Peripheral scatter-gather using the alternate data structure

DMA – Basic DMA transfer

- The peripheral sends a request to the DMA peripheral for <u>each individual</u> DMA transfer
  - Example: UART requesting a byte of data to transfer. The UART will only request another byte when its transmit buffer becomes available
- After the number of arbitrated transfers occur, the peripheral's request will have to re-negotiate for the DMA bus
- The basic DMA transfer will continue until the specified amount of memory has been transferred. At the completion of all the DMA transfers, the DMA engine will generate an interrupt to a call-back routine

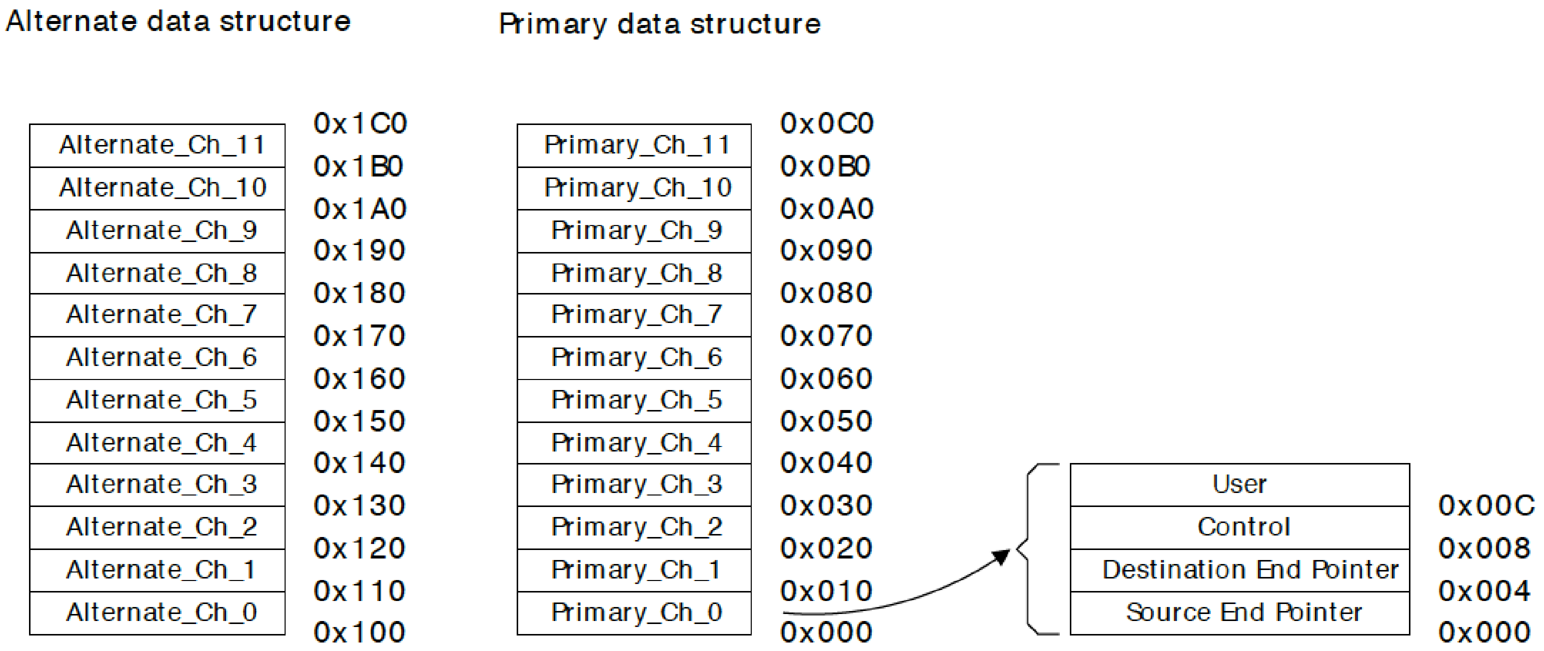
DMA – Auto-request

- Auto-request enables large amount of data to be transferred with a single request to the DMA engine to initiate the DMA transfer
  - Example: Transfer a complete block of memory to NAND
- To continue its DMA operation, it must win the DMA bus for its next DMA arbitrated number of memory transfers. If it loses the DMA bus negotiation, the DMA transfer will hold off until its next DMA bus negotiation cycle to request the DMA bus to continue its DMA transfer
- The auto-request DMA transfer will continue until the specified amount of memory has been transferred. At the completion of all the DMA transfers, the DMA engine will generate an interrupt to a call back routine

DMA - Channel control data structure

- The Channel Control data structure is used to control the DMA request

Figure 8.6. Memory map for 12 channels, including the alternate data structure



DMA – Source End Pointer

User	0x00C
Control	0x008
Destination End Pointer	0x004
Source End Pointer	0x000

- The src_data_end_ptr memory location contains a pointer to the end address of the source data

Table 8.7. src_data_end_ptr bit assignments

Bit	Name	Description
[31:0]	src_data_end_ptr	Pointer to the end address of the source data

DMA – Destination End Pointer

User	0x00C
Control	0x008
Destination End Pointer	0x004
Source End Pointer	0x000

- The dst_data_end_ptr memory location contains a pointer to the end address of the destination data

Table 8.8. dst_data_end_ptr bit assignments

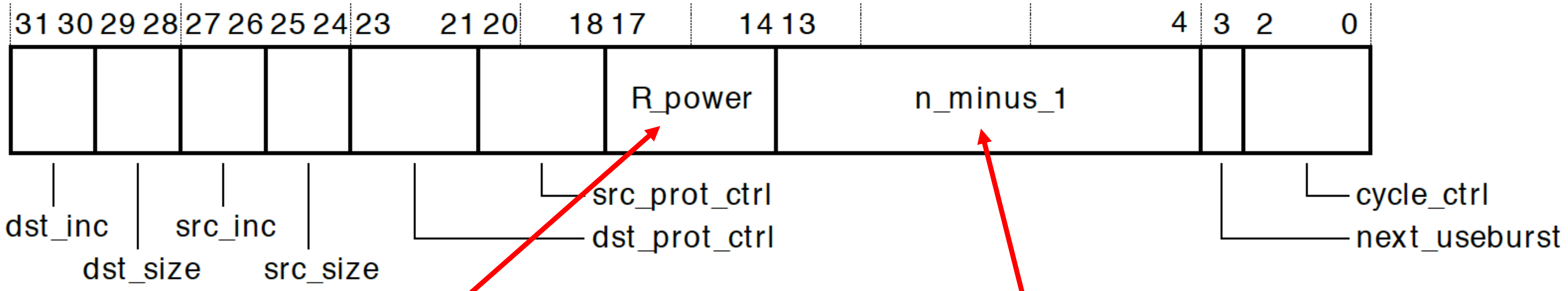
Bit	Name	Description
[31:0]	dst_data_end_ptr	Pointer to the end address of the destination data

DMA – Control

User	0x00C
Control	0x008
Destination End Pointer	0x004
Source End Pointer	0x000

- For each DMA transfer, the channel_cfg memory location provides the control

Figure 8.8. channel_cfg bit assignments



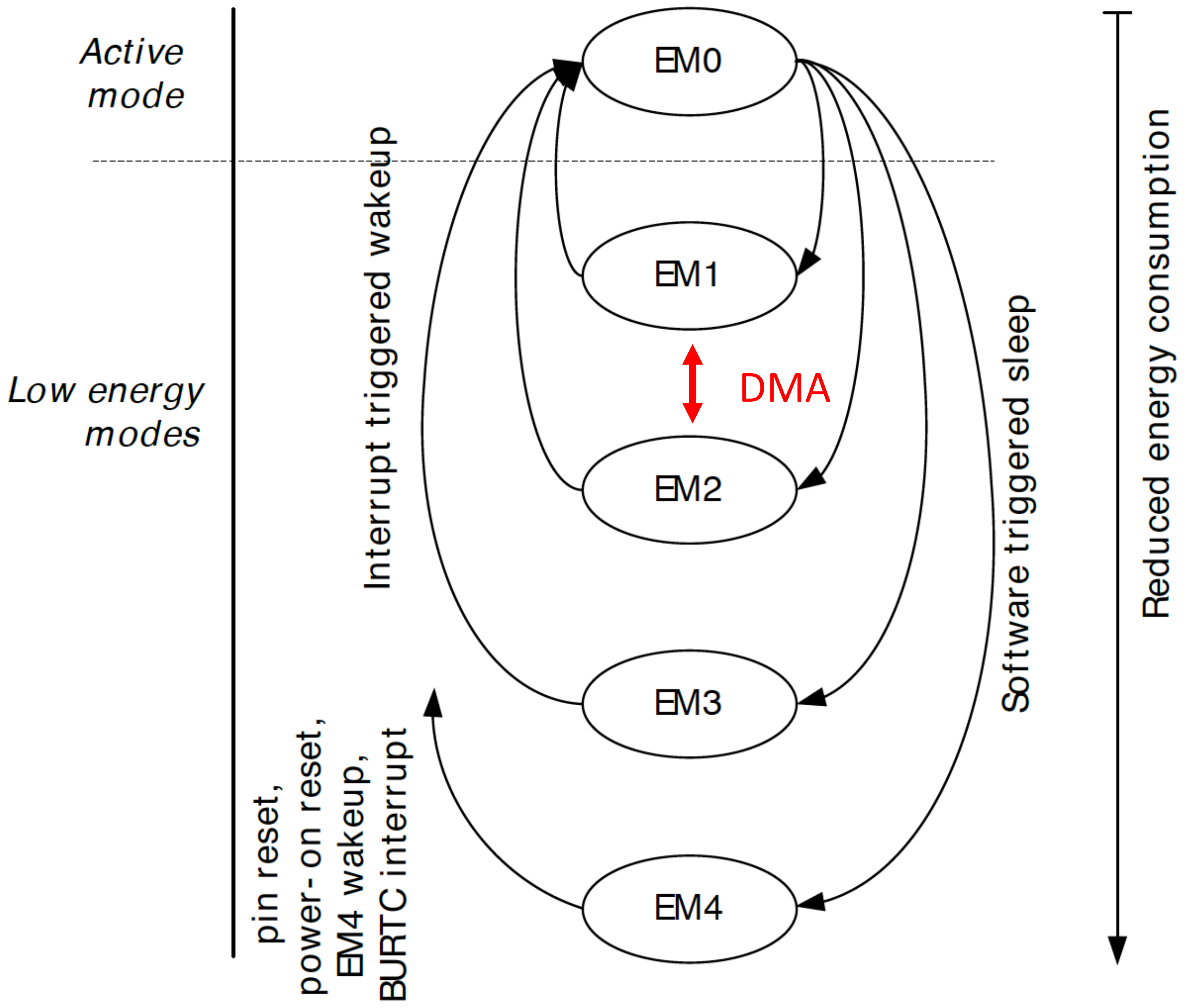
Arbitration

Number of transfers, 10 bits results in 1024 - 1

DMA – EMU interaction

- The DMA unit is the only peripheral that can take the processor into a higher Energy Mode without waking up the CPU.
- The DMA engine can perform a DMA request for an EM2 peripheral like the LEUART0 by momentarily turning on the HFRCO oscillator to perform the DMA transfer. Effectively taking the part to EM1 until the transfer is completed, and then drop back down to EM2.

Figure 10.2. EMU Energy Mode Transitions



DMA - Interrupts

- DMA Interrupts are handled much differently than any other peripheral
- The system utilizes the DMA_IRQHandler to prioritize and respond to the DMA interrupts according
- Instead of using an Interrupt Service Routine, ISR, to handle the end of a DMA operation, a call-back function will be used
- A unique call-back function can be defined per DMA channel which enables specific handling of the end of DMA transferred based on the peripheral
- Upon receiving an interrupt, the DMA interrupt handling routine will call the call-back function to handle the end of the DMA operation

Setting up the DMA

- First, the clock tree to the DMA must be established
  - Without establishing the clock tree, all writes to the DMA registers will not occur
  - DMA clock source is the HFPERCLK, so no oscillator enable is required
    - But, the HFPERCLK does need to be turned on using CMU_ClockEnable
  - Enable the DMA clocking using the CMU_ClockEnable for the DMA

Setting up the DMA

- Second, the DMA must be set up (continue)
  - DMA_Init() to set up the basic operations of the DMA peripheral
  - DMA_CfgChannel() to configure the individual DMA channel
  - DMA_CfgDescr() to configure the DMA descriptors

DMA – DMA_Init();

- DMA_Init();
  - This function will reset and prepare the DMA controller for use. Although it may be used several times, it is normally only used during system init. If reused during normal operation, notice that any ongoing DMA transfers will be aborted. When completed, the DMA controller is in an enabled state.
  - Must be invoked before using the DMA controller.
  - DMA_Init_TypeDef
    - hprot – unit8_t
    - controlBlock – DMA_Descriptor_TypeDef*
      - For this field, use dmaControlBlock

DMA - DMA_CfgChannel()

- DMA_CfgChannel();
  - Configure miscellaneous issues for a DMA channel. This function is typically used once to setup a channel for a certain type of use.
  - DMA_CfgChannel_TypeDef
    - highPri – bool
    - enableInt – bool
    - select – unit32_t (Source of the DMA such as ADC, ACMP, UART, etc.)
    - cb – DMA_CB_TypeDef (Setting up call-back function for this DMA Channel)

DMA - DMA_CfgDescr()

- DMA_CfgDescr();
  - This function is used for configuration of a descriptor for the following DMA cycle types:
    - auto-request - used for memory/memory transfer
    - basic - used for a peripheral/memory transfer
    - ping-pong - used for a ping-pong based peripheral/memory transfer style providing time to refresh one descriptor while the other is in use.
  - DMA_CfgDescr_TypeDef
    - dstInc – DMA_DataInc_TypeDef (how many bytes to increment the destination address)
    - srcInc – DMA_DataInc_TypeDef (how many bytes to increment the source address)
    - size – DMA_DataSize_TypeDef (size of a single DMA tranfer in bytes)
    - arbRate – DMA_ArbiterConfig_TypeDef (how many DMA cycles between arbitration requests)
    - hprot – unit8_t

Setting up the DMA for the ADC

- Summary what is required to configure the ADC for DMA transfers
  - Configure the DMA channel
    - DMA priority
    - DMA request source
    - DMA call back routine
    - DMA_CfgChannel();
  - Configure the DMA descriptors
    - Size of DMA transfers
    - DMA arbitration setting
    - Incrementing of source or destination addresses
    - DMA_CfgDescr();

Setting up the DMA

- Third, the DMA call back function must be written
  - The call back routine will be the DMA's version of the interrupt handling routine
  - Its name can be specified by the programmer and must match the name specified in Channel Configuration setup
  - The call_back variable must be defined as a global variable
  - /* global variable */
    - DMA_CB_TypeDef ADC_cb;
  - /* set up call-back function */
    - cb.cbFunc = ADCdmaTransferDone;
    - cb.userPtr = NULL;
    - cb.primary = true;
  - void ADCdmaTransferDone(unsigned int channel, bool primary, void *user) {
    - Call-back code
    - }

You can have a call-back for each DMA channel

Setting up the DMA

- Forth, the DMA interrupts must be enabled if needed
    - Clear all interrupts from the DMA to remove any interrupts that may have been set up inadvertently by accessing the DMA->IFC register or the emlib routine
    - Enable the desired interrupts by setting the appropriate bits in DMA->IEN
        - The DMA channels that are being used must have their Interrupts Enabled!
    - Set BlockSleep is <u>not</u> set by the DMA peripheral
        - The DMA will utilize the block sleep mode of the peripheral that will be accessing it.
    - Enable interrupts to the CPU by enabling the DMA in the Nested Vector Interrupt Control register using NVIC_EnableIRQ(DMA_IRQn);

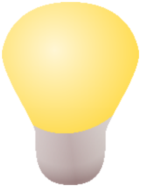
Setting up the DMA

- Fifth, sixth, seventh, ...
  - DMA must be set up before each DMA transfer request can begin or to begin to negotiate for the DMA bus
  - The DMA is set up by calling the particular DMA type of transfer
  - For this course, we will be using DMA_Activate_Basic()

DMA – Setting up the DMA descriptors

- Example: DMA_Active_Basic();
  - **Parameters**
    - [in] **channel** DMA channel to activate DMA cycle for.
    - [in] **primary** true - activate using primary descriptor, false - activate using alternate descriptor
    - [in] **useBurst** The burst feature is only used on peripherals supporting DMA bursts. Bursts must not be used if the total length (as given by nMinus1) is less than the arbitration rate configured for the descriptor. Please refer to the reference manual for further details on burst usage.
    - [in] **dst** Address to start location to transfer data to. If NULL, leave setting in descriptor as is from a previous activation.
    - [in] **src** Address to start location to transfer data from. If NULL, leave setting in descriptor as is from a previous activation.
    - [in] **nMinus1** Number of DMA transfer elements (minus 1) to transfer (<= 1023). The size of the DMA transfer element (1, 2 or 4 bytes) is configured with

Summarizing the steps of initiating a DMA operation



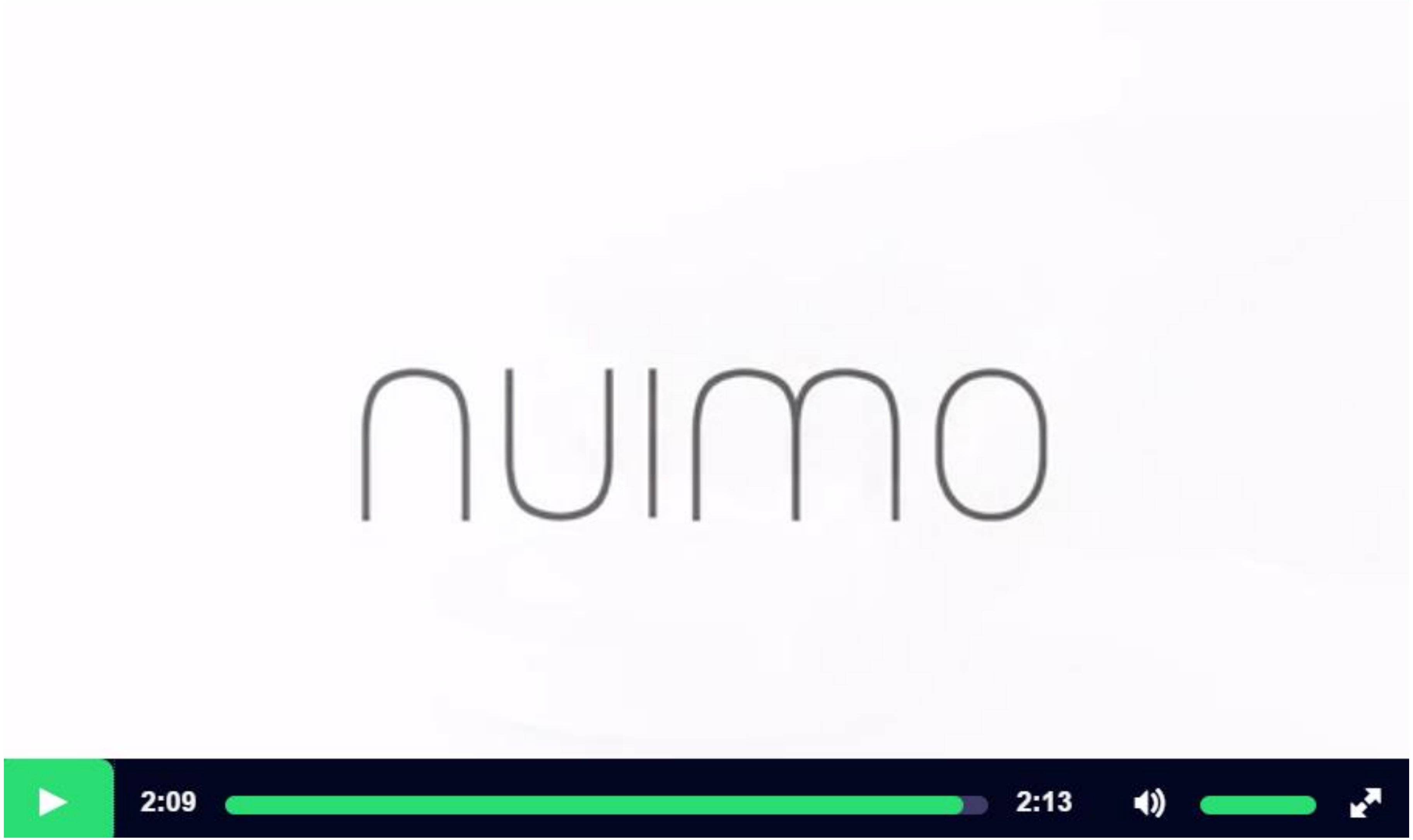
- In the program routine that the DMA transfer will be requested, the following steps are required:
  - Set up the DMA transfer using DMA_Activate_Basic
  - Set the BlockSleep() energy mode of the peripheral that will be requesting the DMA bus
    - Note: It must be EM0 or EM1 unless the peripheral the LEUARTn that can utilize the DMA peripheral in EM2
  - Enable the peripheral that will be utilizing the DMA bus

Summarizing the steps of finish a DMA operation

- In a DMA call back routine, like an like an interrupt handler, basic operations are required:
  - Clear the DMA interrupt register
    - DMA->IFC = DMA channel;
  - Disable or turn off the peripheral
  - Release the blocked energy mode using the unBlockSleep() energy mode of the peripheral that just finished with the DMA bus
    - Note: It must be EM0 or EM1 unless the peripheral the LEUARTn that can utilize the DMA peripheral in EM2

Dmactrl.c

- The Silicon Labs emlib library for the DMA peripheral has been distributed on the slack channel assignment3
- Please insert this library into your /src directory of your projecet
- It will be needed for your assignment #3, and for your future assignment



https://www.kickstarter.com/projects/802159142/nuimo-seamless-smart-home-interface?ref=discovery

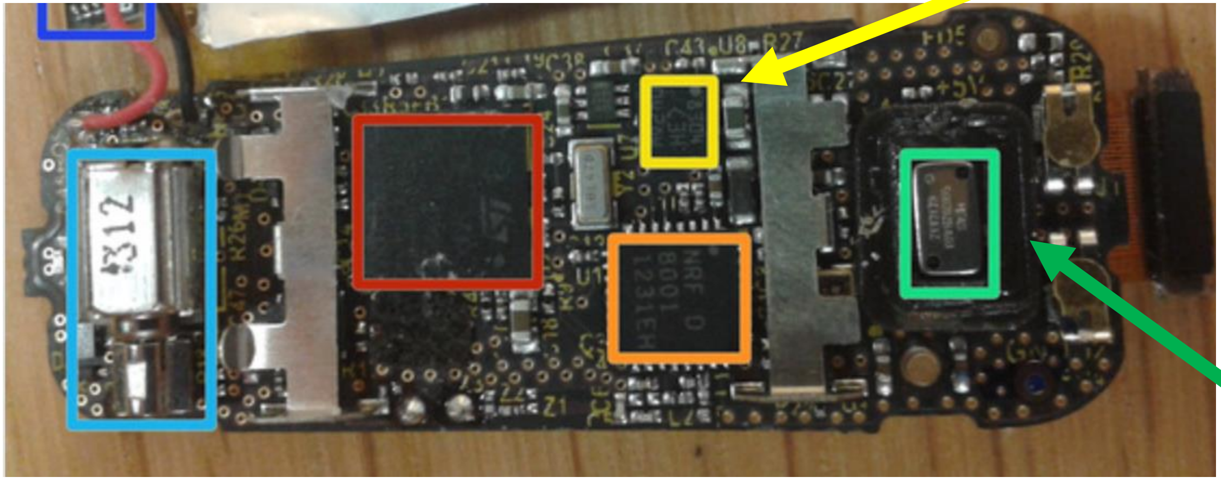
Sensors – For the low energy market

- 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

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

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.)

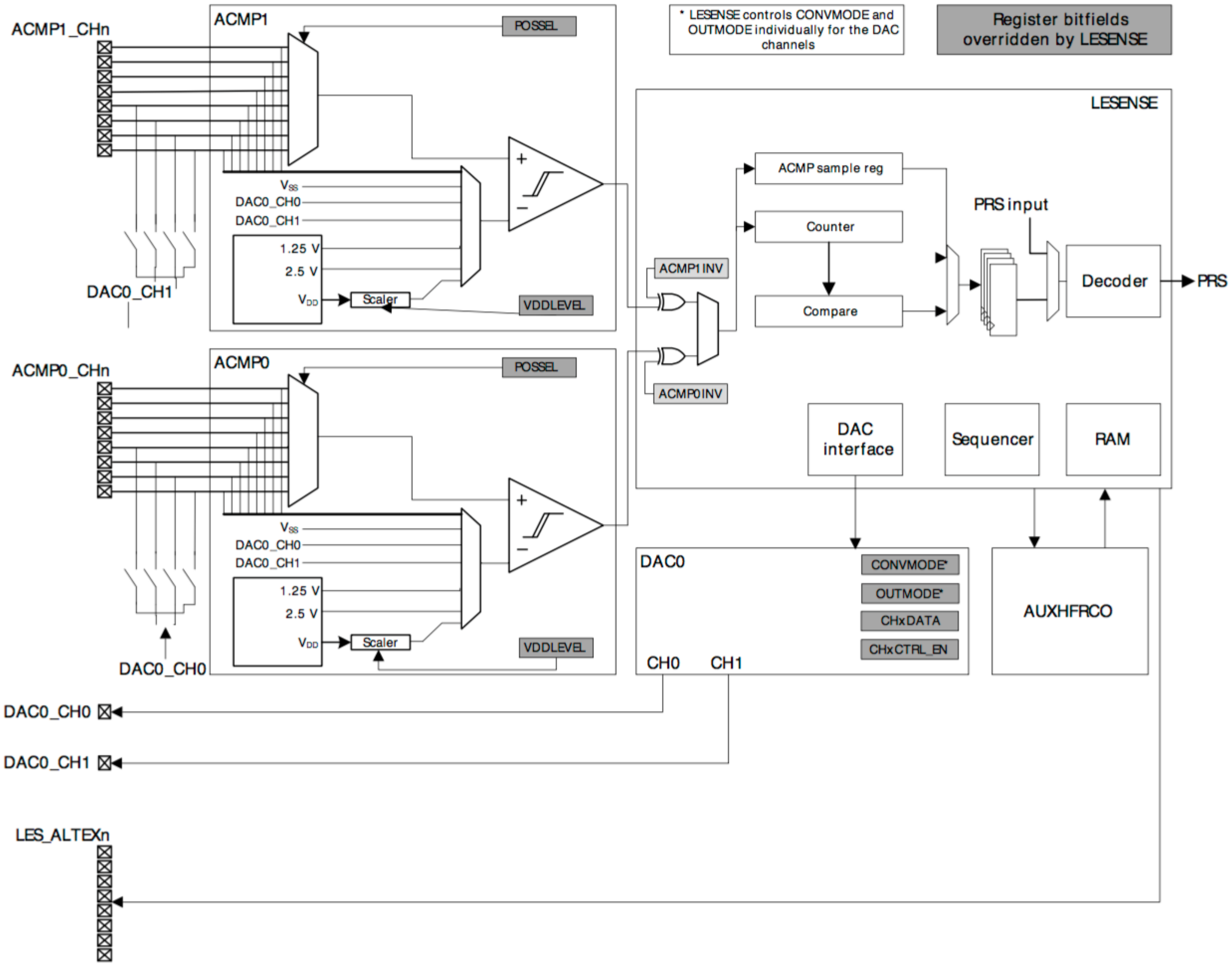
Parameter	Symbol	Conditions	Min	Typ	Max	Units
High Supply Voltage	V_{DD}		2.0	3.3	3.6	V
Low Supply Voltage	V_{REG}		1.71	1.8	2.75	V
Average Supply Current(1)	I_{DD}	Run Mode @ 1 ms sample period		393		µA
		Run Mode @ 2 ms sample period		199		µA
		Run Mode @ 4 ms sample period		102		µA
		Run Mode @ 8 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

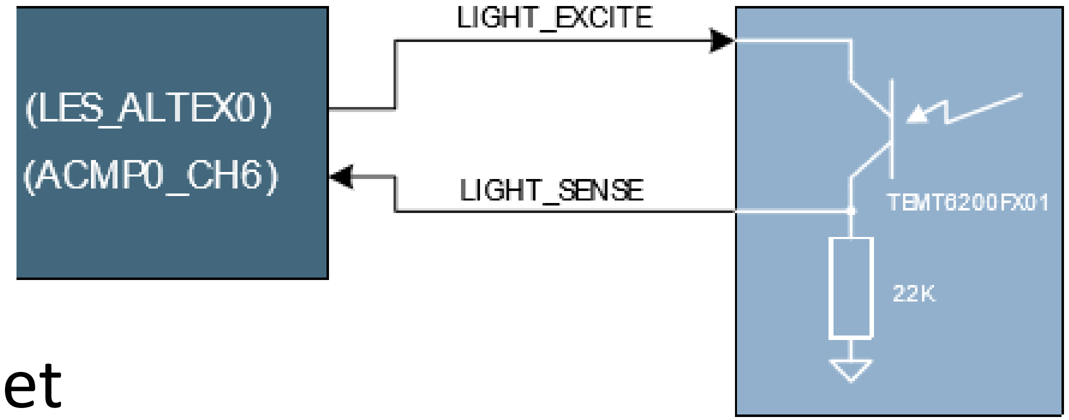
Figure 1.1. LESENSE Overview

LESENSE

- Integrated 16 channel passive sensor state machine to the Silicon Labs' Leopard Gecko.
- Works in EM2 state which is much less energy than many of the active sensor standby currents
- Performance many not be equivalent to the comparative active sensors



Passive Ambient Light Sensor on the Leopard Gecko starter kit



- How much current is required to get light_sense to equal 3.3v?
  - 3.3v (light_excite) / 22Kohms = 0.150mA
- Only an estimation since the photo diode is being powered, Vce, to 3.3v, and not 5.0v
  - 800 lx
- If your want full range of 10,000 lx, the current while sensing would be 2mA approximately
  - The resistor would need to decrease from 22Kohms to 1.65Kohms

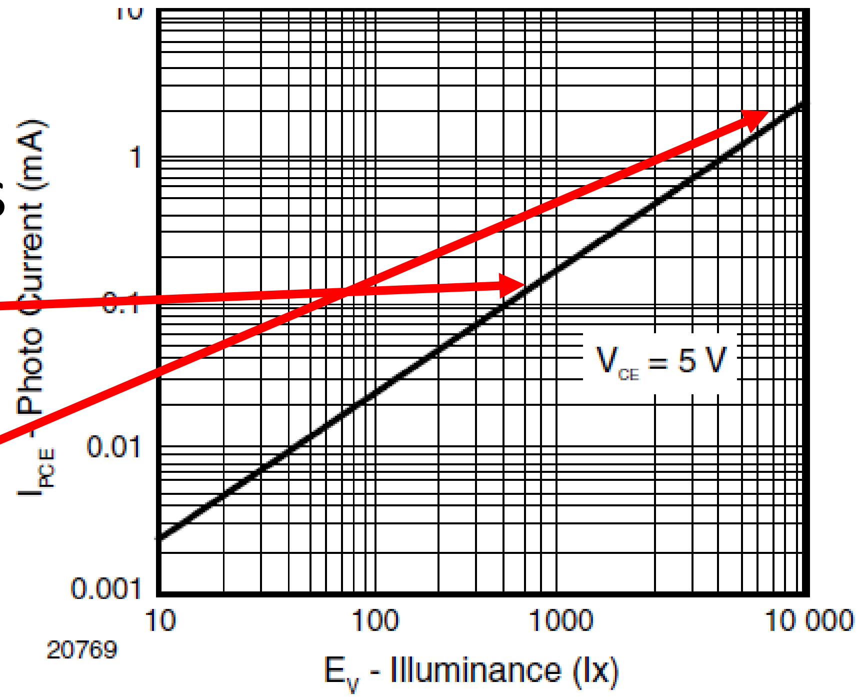


Fig. 4 - Photo Current vs. Illuminance

Active ambient light sensor

- On-Semi LV0104CS I2C active ambient light sensor
  - It can measure lux into the 10,000 lx range as in the previous sample
  - It is lower current in active mode by 1,900uA (2,000 – 100uA)
  - It is higher current in sleep mode by 1uA (1 – 0uA)

LV0104CS

ELECTRICAL AND OPTICAL CHARACTERISTICS at $Ta = 25°C$, $V_{CC} = 2.5V$ (Note 4)

Parameter	Symbol	Conditions	Ratings			Unit
			min	typ	max	
Supply Current	I_{DD}	Ev=0 lux		70	100	μA
Sleep Current	I_{SLP}	Sleep mode, Ev=0 lux			1	μA

When will it be better to implement a 10,000 lux passive ambient light sensor to its active counterpart?

- Assume that the energize time of the passive ambient light sensor is 100uS
- Assume the response time of the On-Semi LV0104CS is 100uS for simplicity
- Active Energy
  - Passive sensor = 2,000uA * 100uS = 200,000u^{2}AS
  - Active sensor = 100uA * 100uS = 10,000u^{2}AS
  - Delta = 190,000u^{2}AS great for the passive sensor
- Passive Energy
  - Passive sensor = 0uA * Tsleep = 0
  - Active sensor = 1uA * Tsleep = 1uA*Tsleep
  - Delta = 1uA*Tsleep u^{2}AS greater for the active sensor
- Solve for Tsleep when both the passive and active sensor have consumed the same amount of energy
  - 1uA*Tsleep = 190,000u^{2}AS
  - Tsleep = 190,000uS or 0.190S

Passive sensor will be lower energy when the sleep time is greater than 190mS!