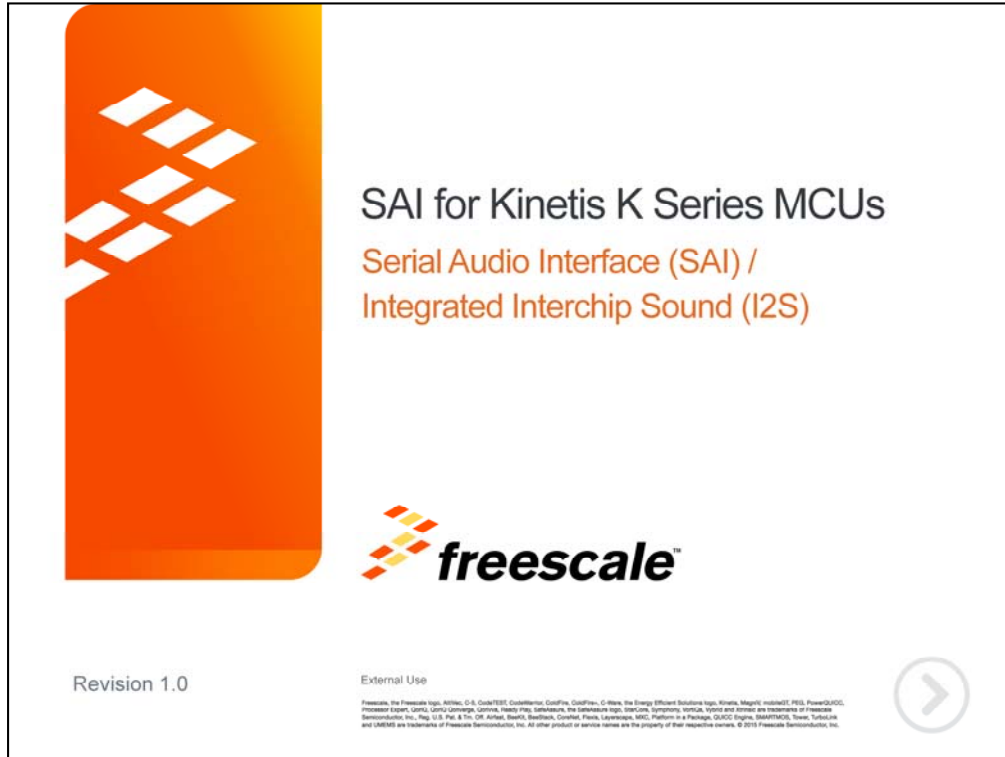




Hello, and welcome to this presentation of the Serial Audio Interface – or SAI – module for Kinetis K series MCUs. You may also have heard SAI referenced as Integrated Interchip Sound -or I-squared-S (I2S). These terms are all used interchangeably.

In this session, you'll learn about the SAI, its main features and the application benefits of leveraging this function.

Agenda

1. Module Overview
2. On-chip Interconnections and Inter-module Dependencies
3. Hardware Configuration
4. Software Configuration
5. Example Use Case
6. SAI Frequently Asked Questions (FAQs)

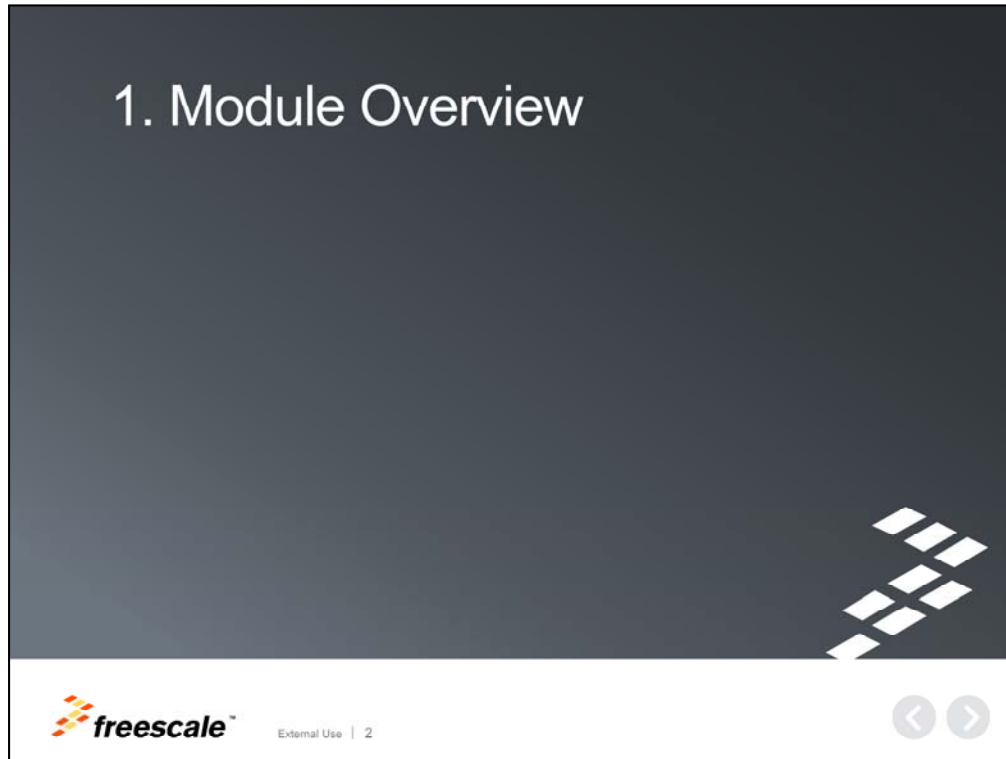


External Use | 1



In this session we'll cover:

- An overview of the SAI module itself
- The on-chip interconnections and inter-module dependencies
- Hardware and software configurations
- An example use case
- And some frequently asked questions



First, let's start with an overview of the module.

SAI Features and Application Benefits

Features

- Receiver and transmitter with independent bit clock and frame sync supporting 2 data channels
- Word size of between 8-bits and 32-bits
- Word size can be configured separately for first word
- 8 Word first in first out (FIFO) for transmit and receive

Application Benefits

- Asynchronous TX/RX can be used for applications that require different playback or recording frequencies
- Reduced pinout with synchronous TX/RX
- Maximum frame size of 32 words for multi-channel audio



External Use | 3



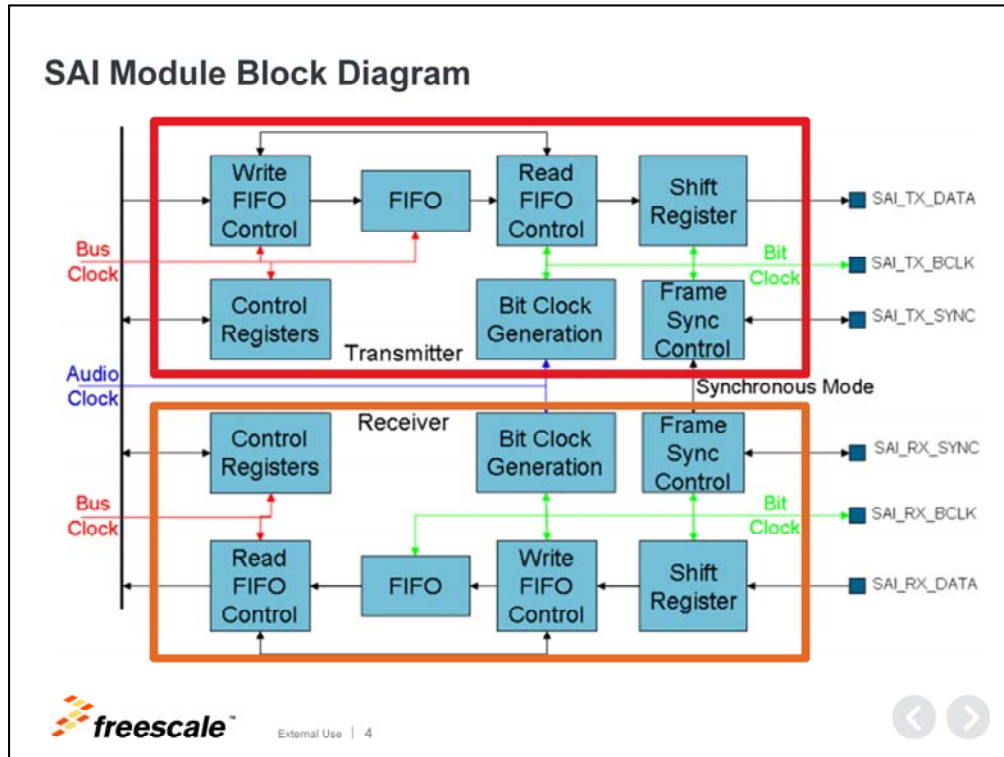
SAI Features and Application Benefits

The SAI module features include:

- A transmitter and receiver supporting up to two data channels with independent clocks. For the exact number of data lines, please refer to the device reference manual.
- The word size, also known as sample size, is variable and can range from 8 to 32 bits
- The word size for the first word can be configured separately from the remaining words in a frame
- The SAI module supports an 8 word first in and first out buffer, or FIFO, which reduces the number of interrupts or module DMA requests. Please refer to the device reference manual for the exact FIFO size.

Application benefits:

- Asynchronous transmitter and receiver, which is useful for applications like a headset implementation requiring audio playback at 48KHz and recording audio at 16KHz
- Synchronous transmitter and receiver reduces pinout for applications with playback and recording at the same sampling frequency
- The SAI module can handle up to 32 words, allowing for multi channel audio implementation

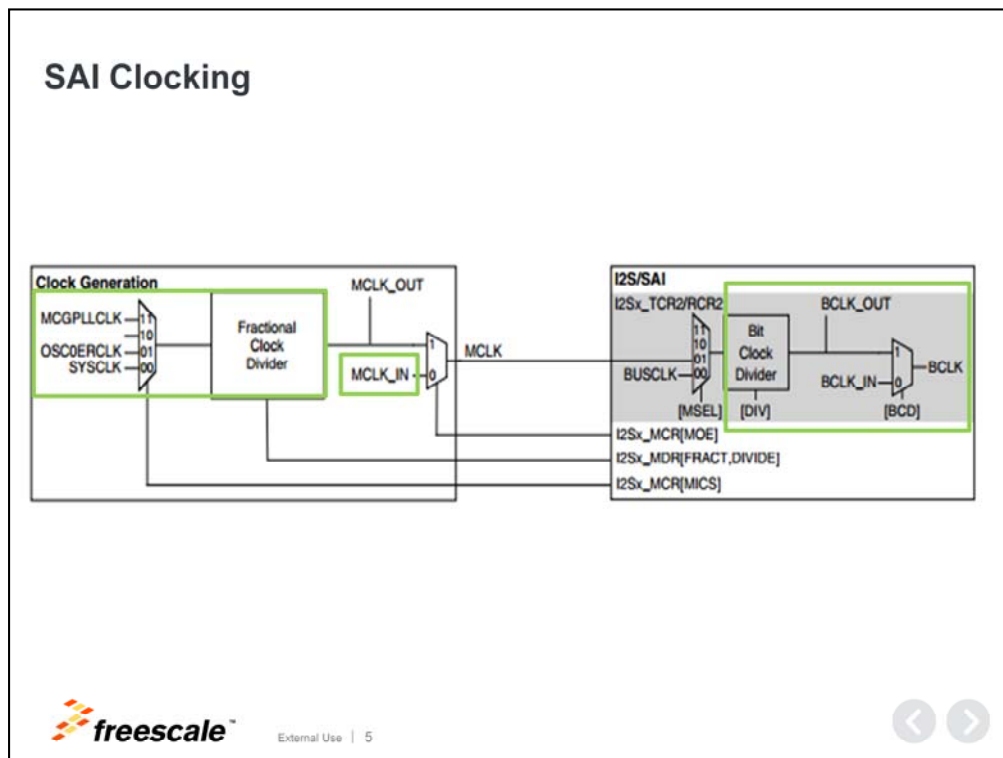


SAI Module Block Diagram

This module has independent clocks for the transmitter and receiver, allowing both synchronous or asynchronous operation.

- When transmitting data using the SAI module, the FIFO holds the words that needs to be sent. The words will be sent whenever a new frame starts. At the start of a new frame, the shift register takes words from the FIFO and starts sending it based on the bit clock.
- When receiving data using the SAI module, at the start of a new frame, the receiver will sample the data received based on the bit clock. The sampled data will then be stored in the FIFO.

I2S is not the only audio protocol that you can use with the SAI; there are variants such as left justified, right justified, PCM, TDM and others.



SAI Clcking

The master clock to the SAI can be internal or external.

For an external clock source, a master clock in (MCLK_IN) signal is used. In case you are using an external clock source, a dedicated oscillator with a frequency multiple of the audio sampling frequency is expected

- When using an internal clock source, the clock must be divided down to generate the MCLK. To achieve this, the SAI implements a fractional divider that takes the clock from one of the clock sources connected to the multiplexer. The MCLK can be shared through a pin if necessary on MCLK_OUT.
- MCLK is used to generate the bit clock and the frame sync.
- When using SAI as a slave, there usually is no need to select anMCLK, because bit clock and frame sync are generated externally. The exception would be if the MCU is an SAI slave and the codec is an SAI master that share a reference clock, in which case the reference clock can be routed through the MCU and out the MCLK pin to the codec.

I2S Requirements

- I2S requires three signals:
 - Frame sync (FS): Used to signal the start of a new frame and possesses the audio sampling frequency
 - Bit clock: Used to clock out/in data
 - Data: Either TX or RX
- Usually a master clock (MCLK) is added for oversampling and as a base clock for the other audio clocks
 - $MCLK = 256 \times FS$ (could also be 384 or 512)



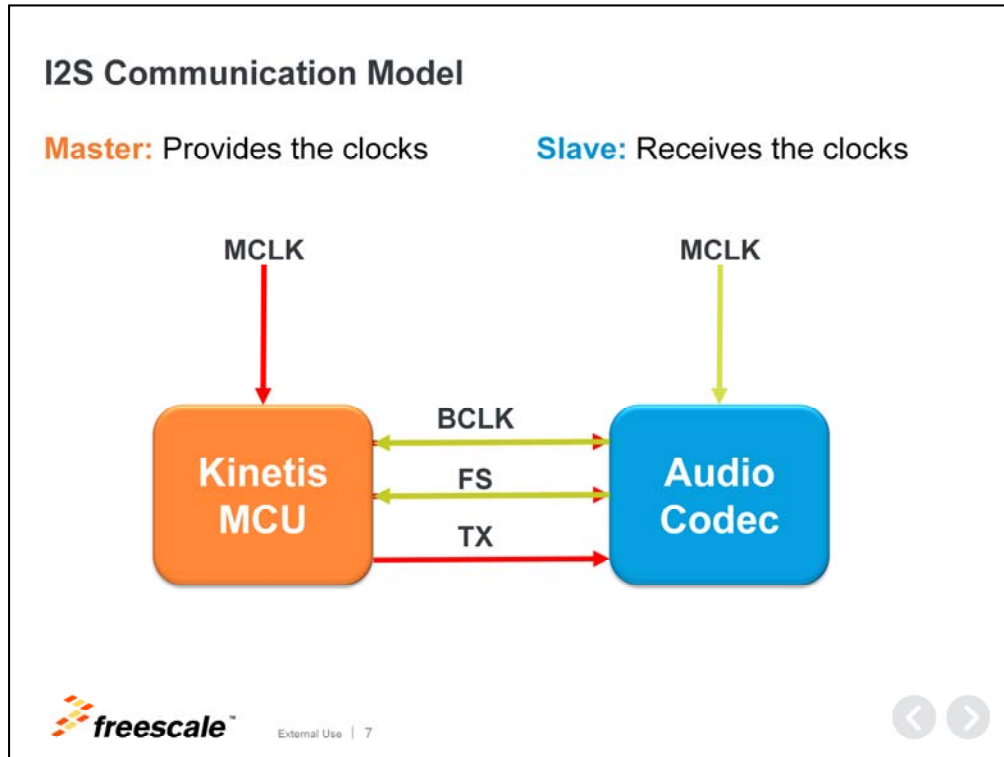
External Use | 6



I2S Requirements

In order to implement the I2S protocol, the following signals are required:

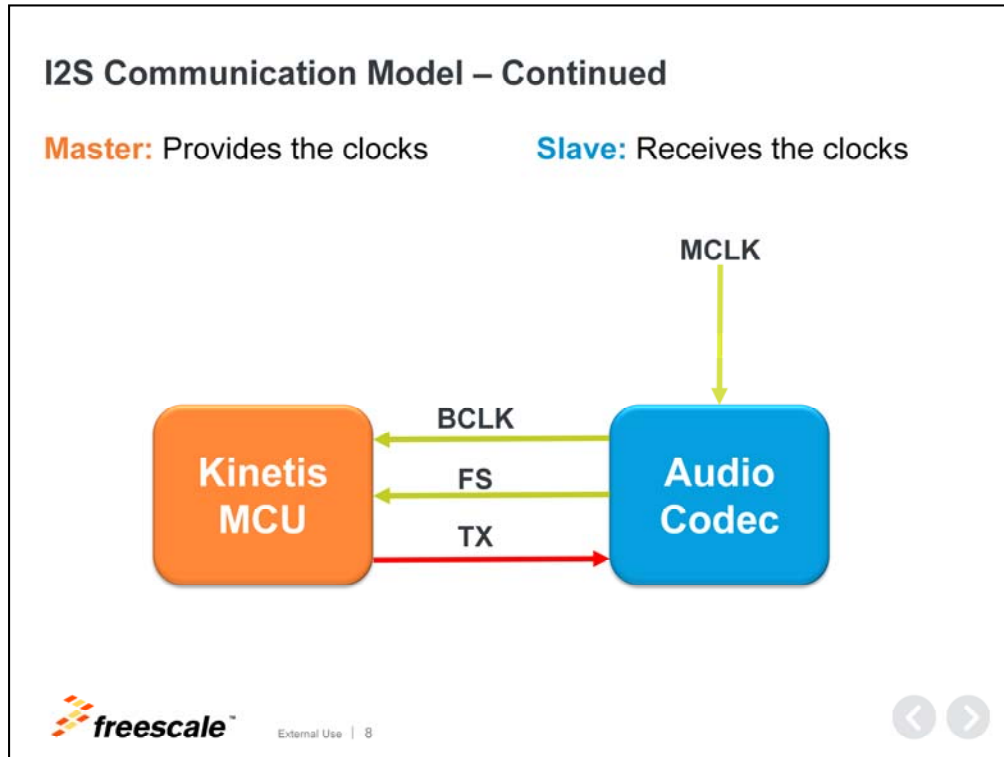
- Frame sync: This signal is required to specify when a new frame of data will start. The frame sync signal is a clock at the audio sampling frequency. The data is sent on the falling edge of the frame sync. You might also hear this signal referenced as frame clock, word select or word clock by other vendors.
- The second signal on the I2S is the bit clock, or sometimes referred to as the serial clock. This clock is used to send or receive the serial data. The speed will depend on the sample size. Assuming that the sample is 32-bits, the serial clock must be 64x the frame sync. Both the frame sync and the bit clock are continuous signals.
- The I2S also requires data lines: TX or RX.
- Finally, all the clocks must use a reference that is called a master clock. This clock can be 256, 384 or 512 times the frame sync.



I2S Communication Model

The I2S communication protocol needs two devices: a master and a slave. The master device provides frame sync and serial clock to the slave.

In this example, the Kinetis MCU is the master and the audio codec is the slave. Note that there's no MCLK going from the master to the slave. The reason is that the MCLK is used to generate the other clocks. This means the master is the only device with MCLK.



I2S Communication Model – Continued

There are some cases where the MCLK must be fed to the slave, but this depends on the overall system configuration.

In this situation, the Kinetis MCU is the slave since it's receiving clocks from the audio codec.

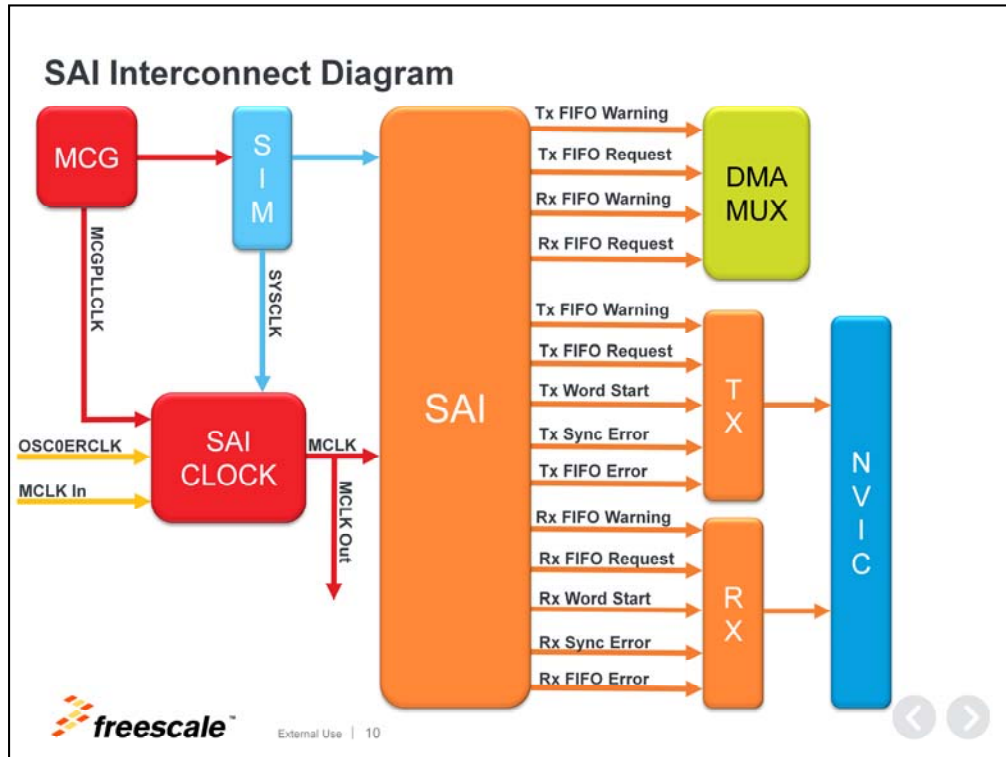
2. On-chip Interconnections and Inter-module Dependencies



External Use | 9



Now, let's talk about on-chip interconnections and inter-module dependencies.



SAI Interconnect Diagram

The diagram here displays the SAI module and its relation to other peripherals.

As previously discussed, the SAI can be connected to different clock options, from internal and external clocks.

The module has several interrupt sources for:

- A FIFO watermark reached
- A FIFO empty or full
- The instance when the FIFO has underrun or overrun
- The instance there's an error with the externally generated frame sync

Transmitter and receiver have independent interrupt vectors.

FIFO watermark and FIFO full or empty can generate a DMA request to offload CPU and fill or empty the FIFO in the background.

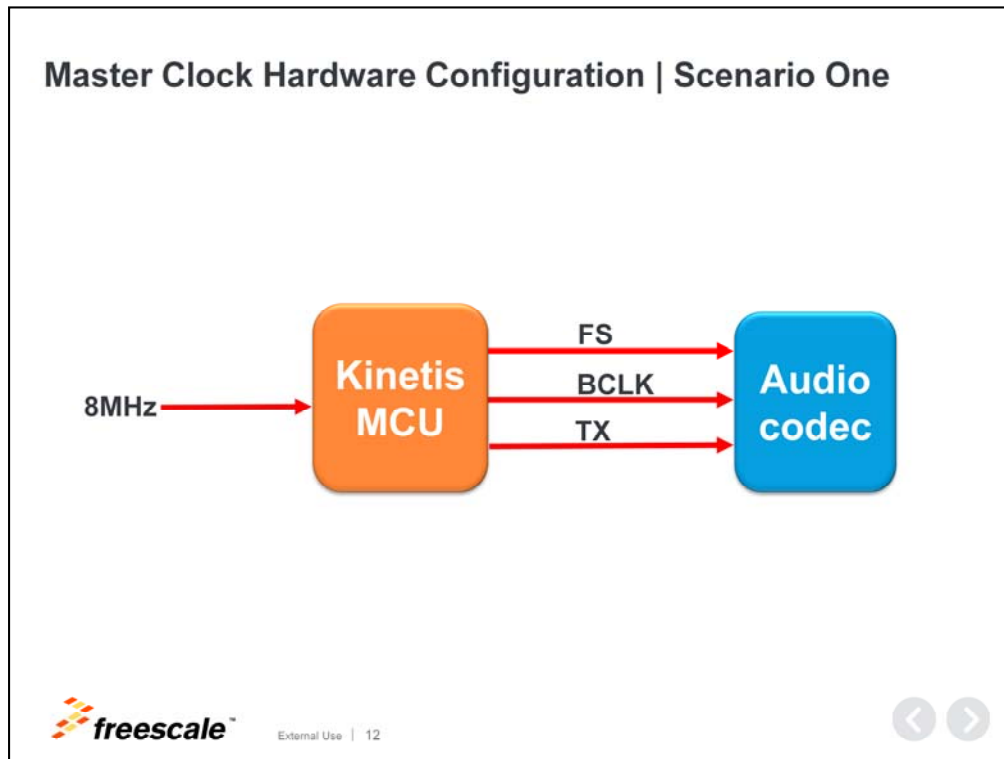
3. Hardware Configuration



External Use | 11



Next, let's go over the hardware configurations of the SAI module.



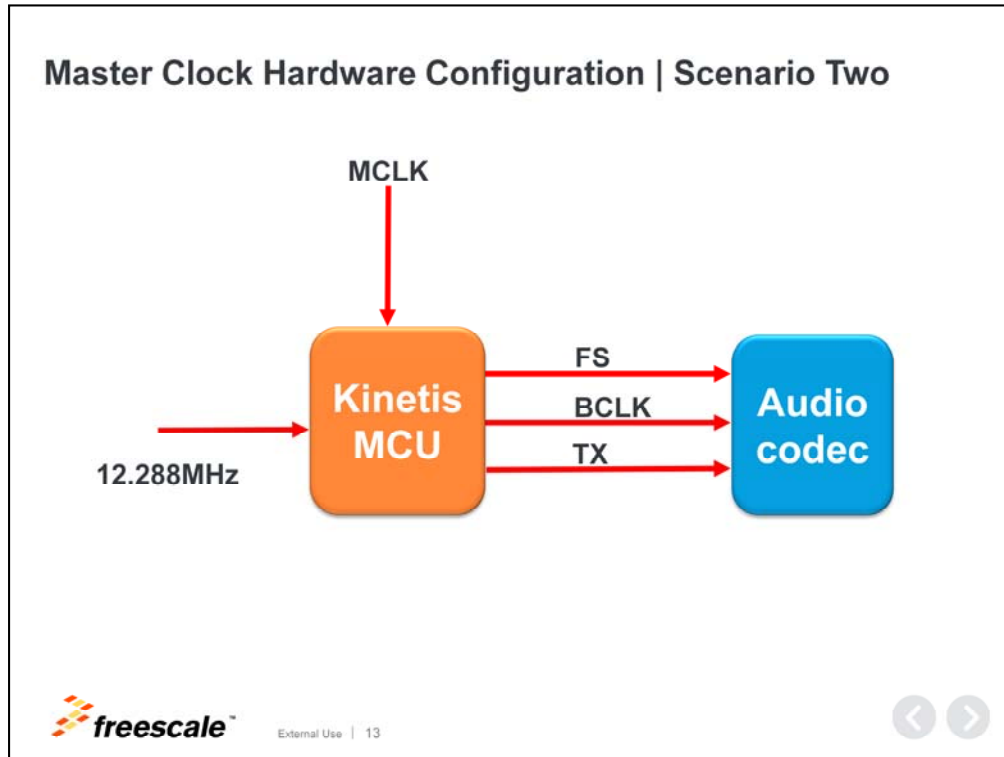
Master Clock Hardware Configuration | Scenario One

The master clock is used to generate the different clocks for I2S communication. The source and values of this clock vary based on the use case.

Let's assume we have a system that plays back sound, which represents the example shown here. A Kinetis MCU and an audio codec are connected through I2S bus and there's an 8Mhz crystal connected to the Kinetis MCU.

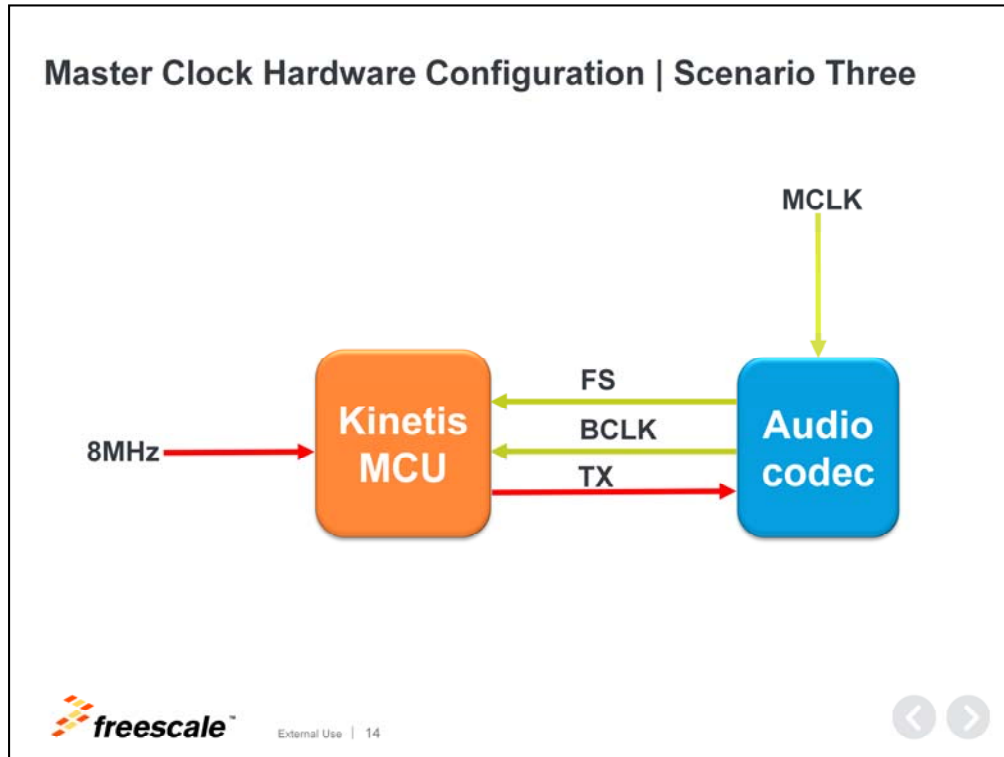
We'll review four different clock connection scenarios.

In the first scenario, the Kinetis MCU is the I2S master and generates the master clock internally. In this use case, just bit clock and frame synch are connected to the audio codec. The advantage for this scenario is that there's no dedicated crystal that reduces the bill of materials (or BOM) cost. The disadvantage is that the jitter generated by the internal fractional dividers might not be suitable for applications where you need high audio quality.



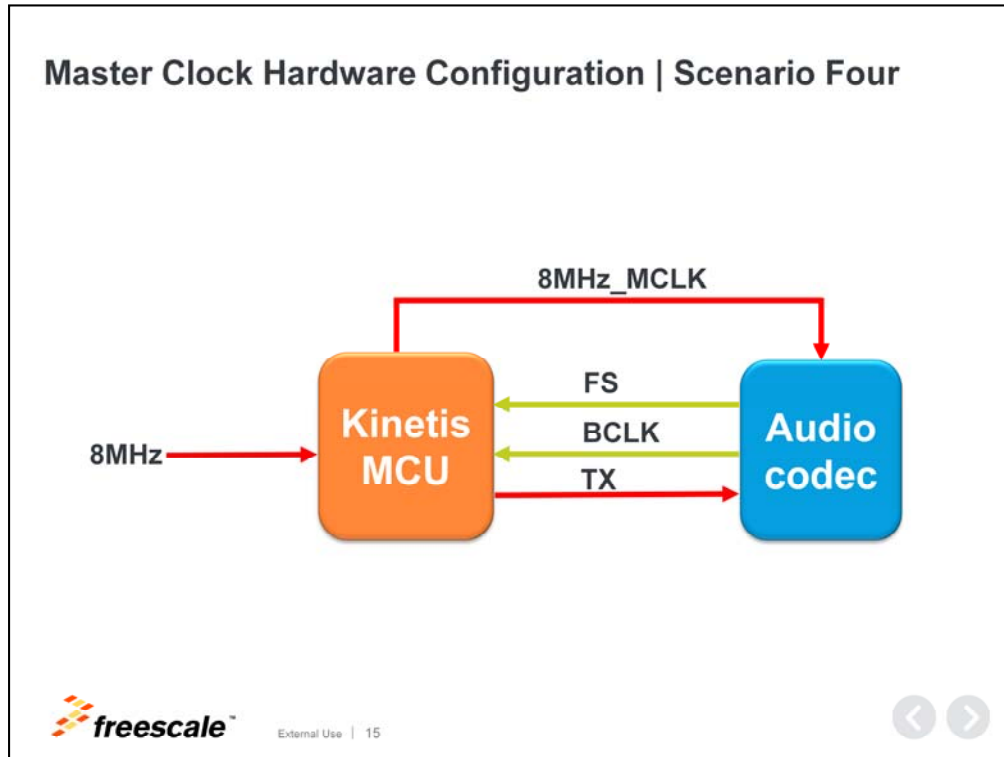
Master Clock Hardware Configuration | Scenario Two

In the second scenario displayed here, the Kinetis MCU is the I2S master, the master clock is sourced from an external oscillator with a specific frequency, let's say, 12.288MHz. For this scenario, the advantage is that the jitter noise is reduced, which is suitable for applications where the audio quality is key. The disadvantage is a more expensive BOM due to the addition of an external crystal.



Master Clock Hardware Configuration | Scenario Three

Scenario three is similar to scenario two, except the Kinetis MCU is the I2S slave and the audio codec is the I2S master.



Master Clock Hardware Configuration | Scenario Four

Lastly, in scenario four displayed here, some audio codecs implement a phase-locked loop, or PLL, tuned for audio applications and capable of generating more accurate I2S clock frequencies. If this feature is available on the audio codec, then the 8Mhz clock from the Kinetis MCU can be used to provide a master clock to the audio codec. In this scenario, the Kinetis MCU remains as the I2S slave and the audio codec is the master. The advantages for this scenario are a lower BOM cost and reduced jitter because the audio codec PLL is created for this type of use case.

4. Software Configuration



External Use | 16



In this next section we'll discuss software configuration.

SAI Kinetis SDK Initialization Example

```

#define SAI_INSTANCE(0)

sai_user_config_t tx_config;
sai_data_format_t format;
sai_state_t tx_state;

//Initialize configuration structure.
tx_config.protocol = kSaiBusI2SType;
tx_config.channel = 0;
tx_config.slave_master = kSaiMaster;
tx_config.sync_mode = kSaiModeAsync;
tx_config.bclk_source = kSaiBclSourceMcLkDiv;
tx_config.mclk_source = kSaiMcLkSourceSysclb;
tx_config.watermark = 4;

//Data format of audio data
format.bits = 16;
format.sample_rate = 44100;
format.mclk = 256 * format.sample_rate;
format.mono_stereo = kSaiStereo;
//Initialize SAI Tx.
SAI_DRV_TxInit(SAI_INSTANCE, &tx_config, &tx_state);
//Configure the data format of SAI Tx.
SAI_DRV_TxConfigDataFormat(SAI_INSTANCE, &format);
//option: register callback functions for finished transfer
SAI_DRV_TxRegisterCallback(SAI_INSTANCE, callback, callback_param);
//start send data
SAI_DRV_SendDataInt(SAI_INSTANCE, addr, len);

-
//Stop Tx.
SAI_DRV_TxStopModule(SAI_INSTANCE);
//De-initialize Tx.
SAI_DRV_TxDeinit(SAI_INSTANCE);

```



External Use | 17



SAI Kinetis SDK Initialization Example

The Kinetis SDK integrates an SAI peripheral driver that can be used to output I2S data.

First, a configuration structure must be filled with:

- The SAI bus configuration
- Whether it will be a master or slave, and
- The references that will be used for the clocks and FIFO watermark

After that, an audio format structure is filled with the actual audio requirements. In the example displayed here, we use 16-bits per sample, with a 44.1 KHz sampling rate. The master clock generated by SAI will be 256 times the sampling frequency and we want a stereo output.

Once the structures are filled, call the API to initialize the module, register a callback for receiving notifications and finally, send data as required.

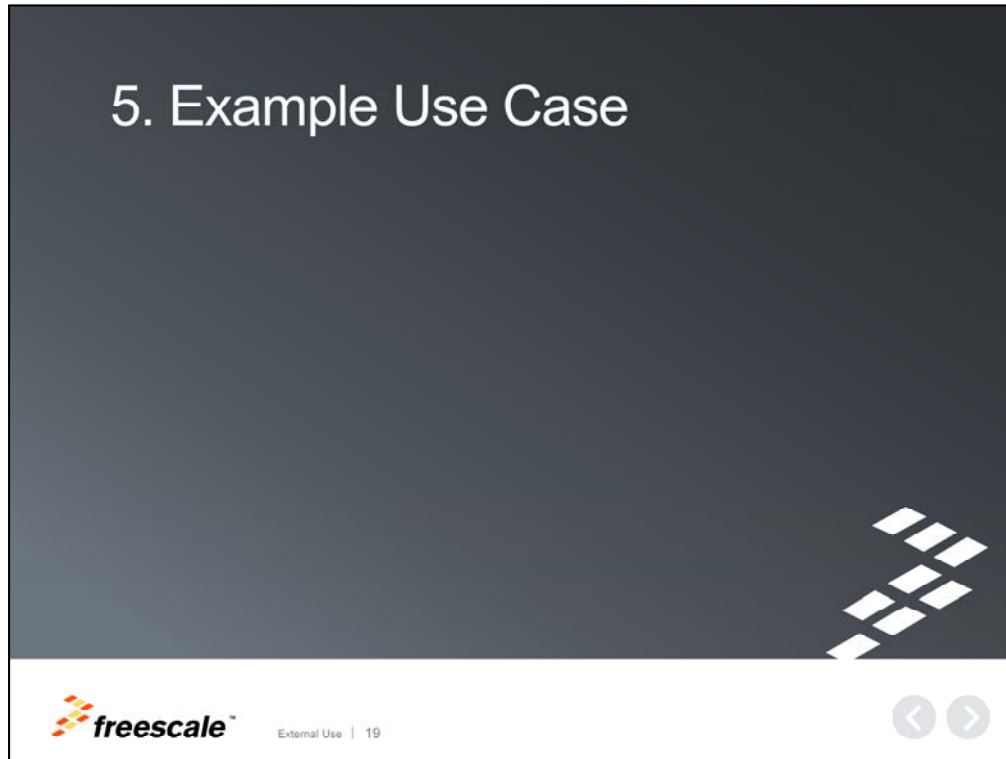
Once the application is done sending data, the SAI transmitter can be stopped, and if required, the peripheral driver can be de-initialized.

SAI Operation In Low-Power Modes

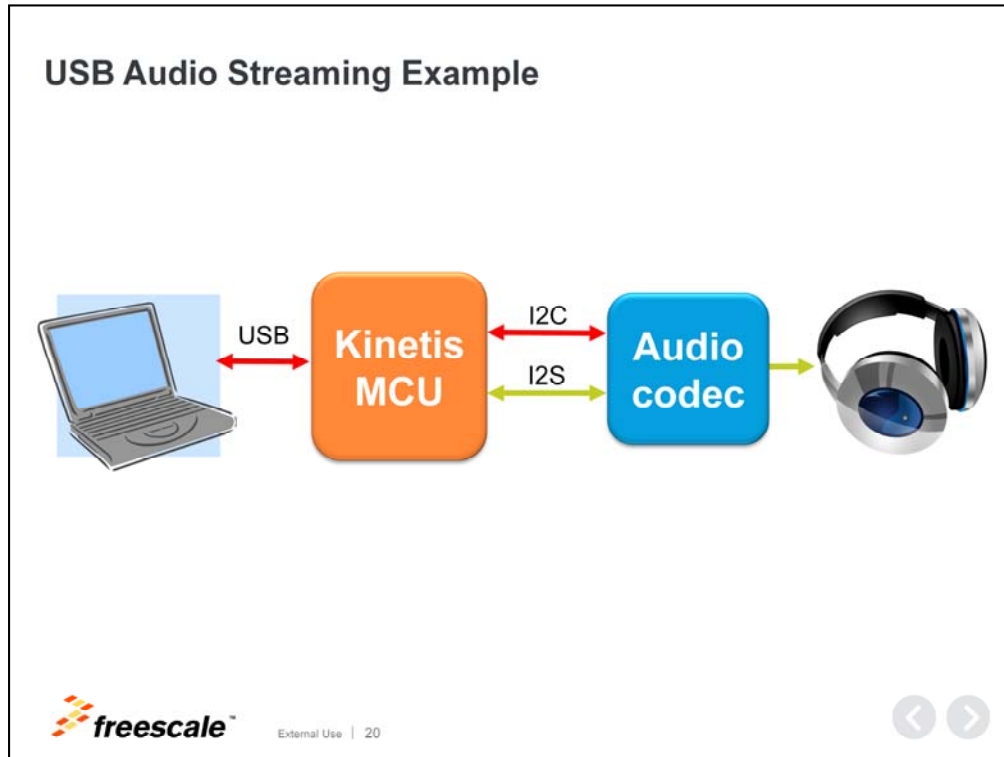
Low-Power Mode	SAI Operation	Comments
STOP	Fully functional with external clock	Use an externally generated bit clock or an externally generated audio master clock (including EXTAL)
VLPR	Fully functional with external clock	Frequency is limited in VLPR mode
VLPW	Fully functional with external clock	Frequency is limited in VLPW mode
VLPS	Fully functional with external clock	Use an externally generated bit clock or an externally generated audio master clock (including EXTAL)
LLS	Static	Registers will retain state
VLLSx	Off	I2S state is not retained

SAI Operation In Low-Power Modes

For battery powered applications, the SAI can be used in different low-power modes in slave or master mode using an external source for the master clock. For low leakage stop modes, the SAI module is static.



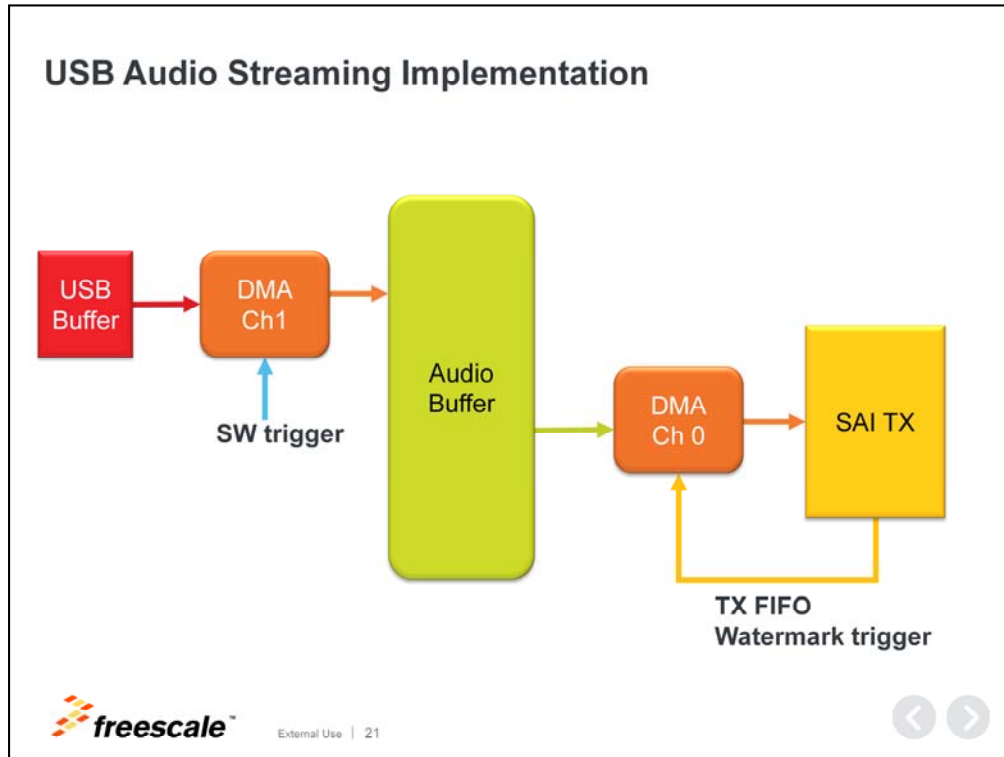
Now, let's talk review an example use case.



USB Audio Streaming Example

Here, we have a USB audio streaming example, and let's say we want to develop a USB speaker using Kinetis MCUs.

- For this application, we'll need a Kinetis MCU with USB support to communicate to a PC.
- An audio codec is required to translate the audio data into sound on your speaker.
- The Kinetis MCU will need to configure the audio codec, and let's assume the communication between these devices is I2C.
- Finally, for transferring audio data from the MCU to the audio codec, I2S is used.



USB Audio Streaming Implementation

In this example, the audio data is received from USB into a buffer. We can use a DMA channel to take this data and move it to a buffer dedicated for the audio samples.

A second DMA channel can be used to move the data from the audio buffer to the SAI TX FIFO. This DMA will be triggered by the SAI TX FIFO watermark signal.

With this use case, the CPU can handle other tasks such as audio synchronization. There is no need to send audio data to the audio codec, since everything is done in the background.

6. SAI Frequently Asked Questions (FAQs)



External Use | 22



Finally, some frequently asked questions about the SAI module:

SAI FAQs

Q: My audio codec has WS, SCK and TX data; how can I connect it to a Kinetis device?

A: There are many names for I2S lines. Below is a table showing the names you might see. The MCU line names may have a TX or RX prefix depending on which line is the asynchronous one.

Device\Line Name	Master clock	Bit Clock	Frame synch	Data transmission	Data receive
Kinetis MCU	MCLK	TX_BCLK /RX_BCLK	TX_FS / RX_FS	TXDx	RXDx
SGTL5000	MCLK	SCLK	LRCLK	DOUT	DIN
DA7321	MCLK	BCLK	WCLK	DI	DO
AK4954A	MCKI	BICK	LRCK	SDTO	SDTI



External Use | 23



SAI FAQs

Q: My audio codec has WS, SCK and TX data; how can I connect it to a Kinetis device?

A: There are different names for the I2S lines, which might be confusing when trying to interface an audio codec with a Kinetis device. The table here shows a few examples on the different line names and how they relate to Kinetis devices.

SAI FAQs – Continued

Q: Is the SAI capable of multichannel audio?

A: Yes, the SAI module is capable of multichannel audio using either Time Division Multiplexing (TDM) method when all the data is sent through the same TX line, or by using different TX or RX lines assigned to different channels.



External Use | 24



SAI FAQs – Continued

Q: Is SAI capable of multichannel audio?

A: Yes, the SAI module is capable of multichannel communication either by using time division multiplexing, which is a method that sends all channel data on the same data line, or by assigning different transmit or receive lines to different channels.

Time division multiplexing may also be referred to as “network mode.”

References

- **App notes:**

- [AN4800](#) An I2S (Integrated Interchip Sound Bus) Application on Kinetis
- [AN4369](#) Audio Output Options for Kinetis

- **Website:** Freescale.com/Kinetis

- **Community:** community.freescale.com/community/Kinetis



External Use | 25



This concludes our presentation on the SAI Module for Kinetis K series MCUs.

More information about the SAI module, can be found in the application notes listed here.

We also invite you to visit us on the web at Freescale.com/Kinetis and check out our Kinetis community page.



www.Freescale.com

© 2015 Freescale Semiconductor, Inc. | *External Use*