TB3218

Getting Started with CCL

Introduction

Author: Cristian Pop, Microchip Technology Inc.

The Configurable Custom Logic (CCL) is a programmable logic peripheral that allows the on-chip creation of the logic functions for Microchip tinyAVR® 0- and 1-series, and megaAVR® 0-series devices. The CCL provides programmable, combinational and sequential logic that operates independently of the CPU execution. It can be connected to a wide range of internal and external inputs such as device pins, events, or other internal peripherals and can serve as a "glue logic" between the device peripherals and external devices. This technical brief explains how to use the CCL in order to implement the following functions:

- **Logic AND Gate:**
  Uses CCL to implement a simple logic AND gate
- **State Decoder:**
  Shows how to use CCL combinational logic to detect a specific state of the external signals
- **SR Latch:**
  Uses internal CCL sequential logic to create an SR latch
- **Manchester Encoder:**
  Demonstrates how to use CCL in combination with other peripherals to implement a Manchester encoder

Note: The code examples were developed using the ATmega4809 Xplained Pro board (ATMEGA4809-XPRO).

Table of Contents

1. Relevant Devices

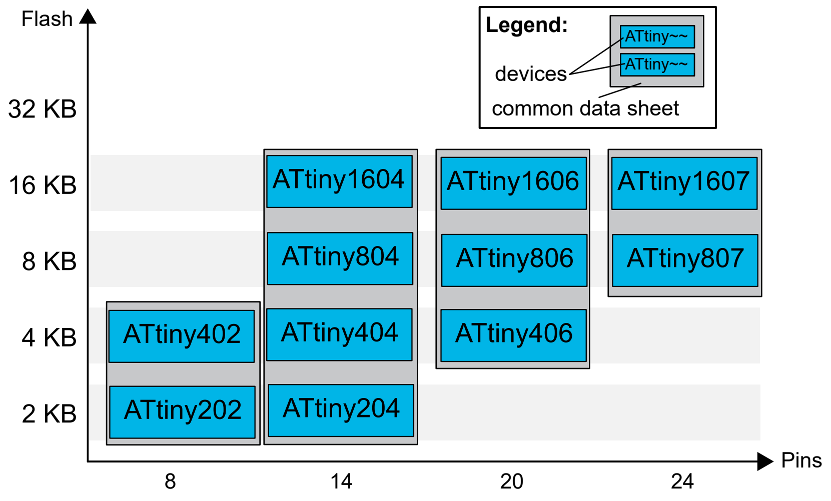
This chapter lists the relevant devices for this document.

1.1 tinyAVR® 0-series

The figure below shows the tinyAVR 0-series, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin- and feature compatible.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.

Figure 1-1. tinyAVR® 0-series Overview



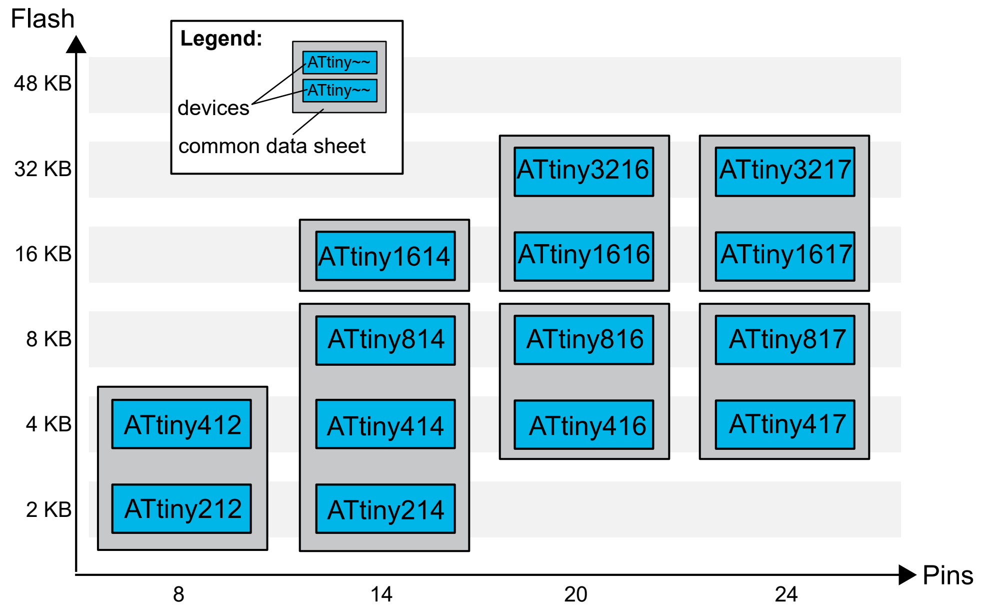
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.2 tinyAVR® 1-series

The following figure shows the tinyAVR 1-series devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin compatible and provide the same or more features. Downward migration may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.

Figure 1-2. tinyAVR® 1-series Overview



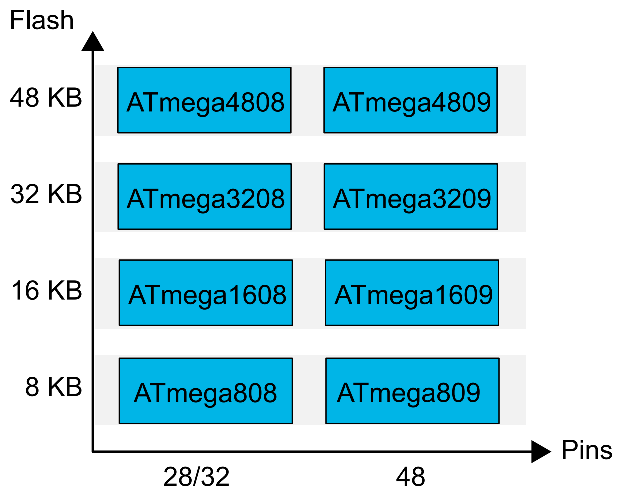
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-series

The figure below shows the megaAVR 0-series devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible.
- Horizontal migration to the left reduces the pin count and, therefore, the available features.

Figure 1-3. megaAVR® 0-series Overview



Devices with different Flash memory size typically also have different SRAM and EEPROM.

2. Overview

The CCL peripheral has one pair of Look-Up Tables (LUTs). Each LUT consists of three inputs with a Truth table, a synchronizer, a filter, and an edge detector. Each LUT can generate an output as a user programmable logic expression with three inputs; any device with CCL will have a minimum of two LUTs available. These inputs can be individually masked. The output can be generated from the combinatorial inputs and be filtered to remove spikes. An optional sequential logic module can be enabled. The inputs to the sequential module are individually controlled by two independent, adjacent LUT outputs (LUT0/ LUT1), enabling complex waveform generation.

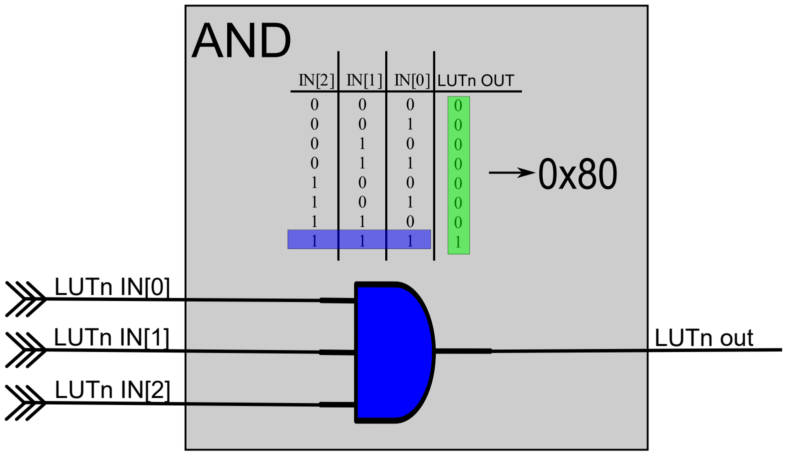
Truth Table

It is possible to create simple logical blocks (AND, OR, NAND, NOR, XOR) or custom ones using the truth table up to three inputs on each of the LUTs. When more than three inputs are required, multiple connected LUTs are used to create logical gates.

To define a combinational specific logic function, the CCL module uses truth tables. A truth table shows how the logic circuit responds to various combinations of three inputs. Each combination of the Input bits (IN[2:0]) corresponds to one bit in the respective TRUTHn register.

Below are some examples on how to create some common logical gates using three inputs.

Figure 2-1. AND Logic

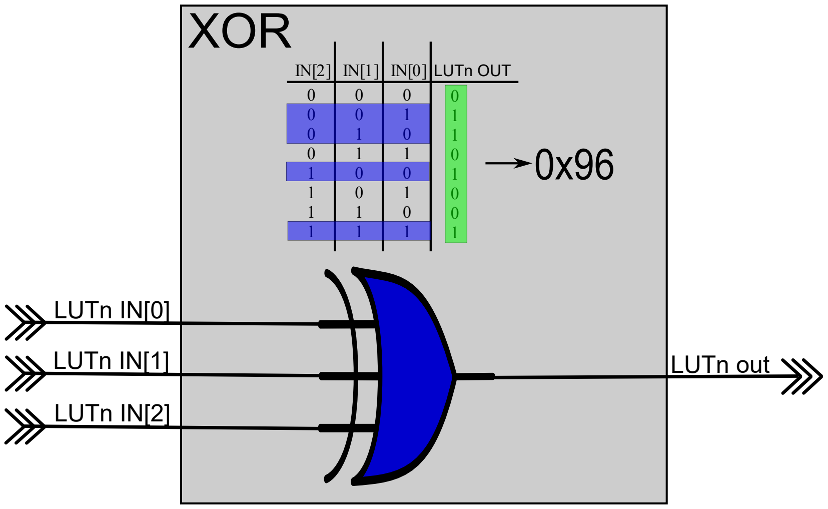


IN[2]	IN[1]	IN[0]	LUTn OUT
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

To get a HIGH(1) output from an AND gate, all inputs must be HIGH(1). Looking at the truth table above, only TRUTH[7] fulfills this requirement if all three inputs are used. This means that TRUTH[7] must be HIGH(1) and the rest must be LOW(0), resulting in the hex value of 0x80 to be used in the TRUTHn register.

```
CCL.TRUTHn = 0x80;
```

Figure 2-2. XOR Gate



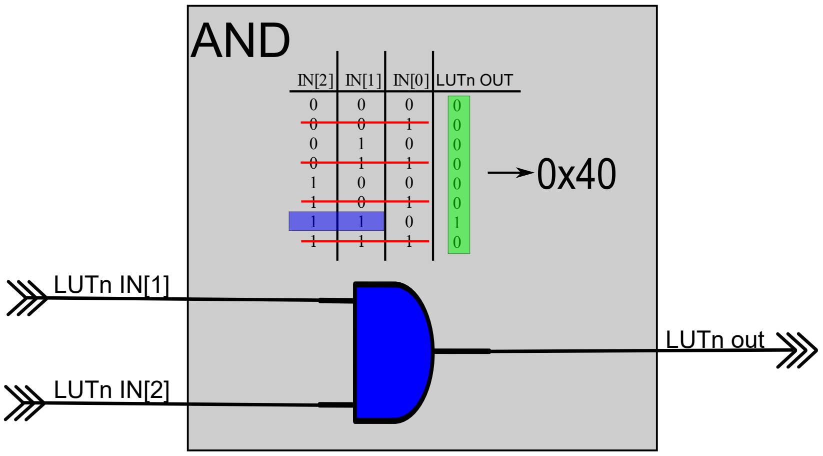
IN[2]	IN[1]	IN[0]	LUTn OUT
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

To get a HIGH(1) output from an XOR gate, the number of HIGH(1) inputs must be odd. Looking at the truth table above, TRUTH[1], TRUTH[2], TRUTH[4], and TRUTH[7] fulfill this requirement. This means that these bits must be HIGH(1) and the rest must be LOW(0), resulting in the hex value of 0x96 to be used in the TRUTHn register.

```
CCL.TRUTHn = 0x96;
```

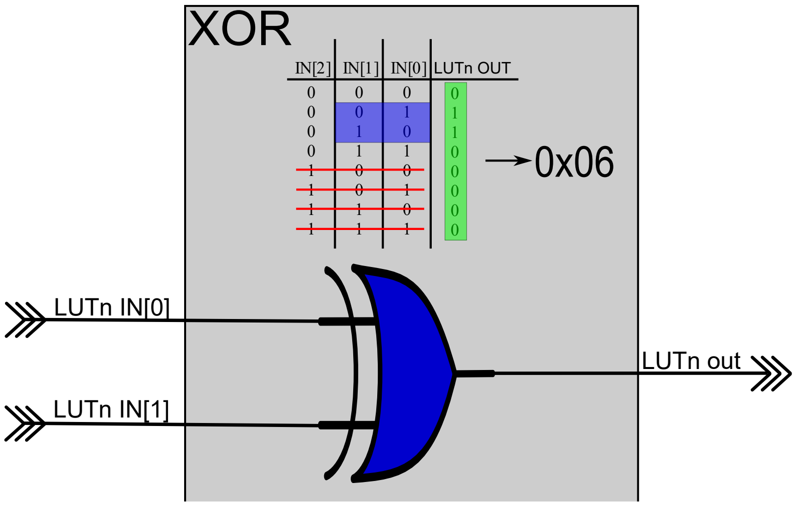
When any of the three inputs are not needed, the unused input will be masked (tied low). Only the TRUTH bits where the masked input is '0' can be used when looking at the truth table to determine how the bits should be set to get the wanted logic. Below are some examples of where various inputs are masked.

Figure 2-3. Two-Input AND Gate, IN[0] Masked



IN[2]	IN[1]	IN[0]	LUTn OUT
0	0	0	0
~~0~~	~~0~~	~~1~~	0
0	1	0	0
~~0~~	~~1~~	~~1~~	0
1	0	0	0
~~1~~	~~0~~	~~1~~	0
1	1	0	1
~~1~~	~~1~~	~~1~~	0

Figure 2-4. Two-Input XOR Gate, IN[2] Masked



IN[2]	IN[1]	IN[0]	LUTn OUT
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
~~1~~	~~0~~	~~0~~	0
~~1~~	~~0~~	~~1~~	0
~~1~~	~~1~~	~~0~~	0
~~1~~	~~1~~	~~1~~	0

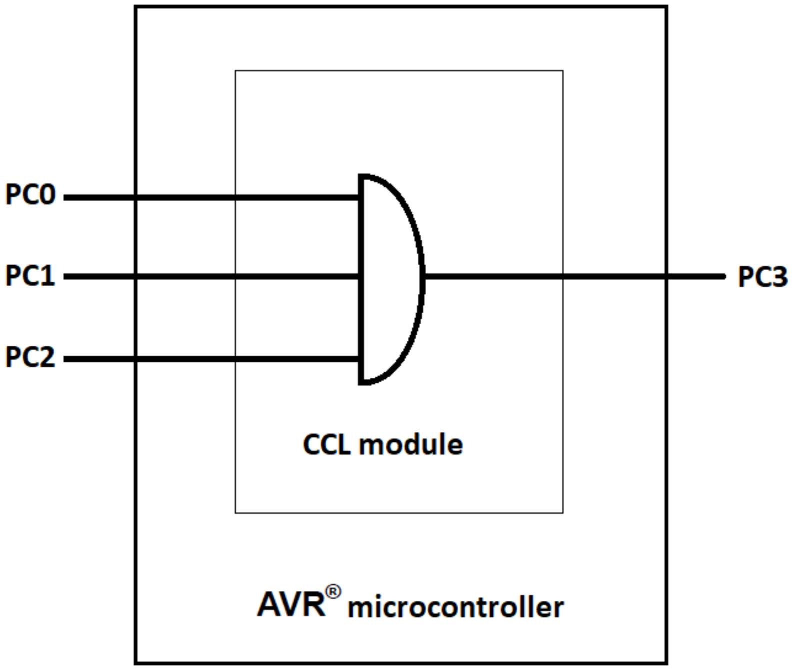
Some applications require more than three logic inputs. The CCL module provides the option to link internally the next LUTs direct output to a LUT input. For example, if LUT0 and LUT1 are used to create a logic function, the LUT1 output can be connected to the LUT0 input internally.

Using the CCL eliminates the need for external logic, reduces Bill of Materials (BOM) cost and enables the CPU to handle time critical parts of the application more efficient.

3. Logic AND Gate

The CCL module can be used to implement a logic gate with up to three inputs. The following example shows how to configure and use CCL LUT1 to implement an AND gate.

Figure 3-1. Using CCL as Logic AND Gate



On the first step, the I/O pins are selected as inputs using the INSELx[3:0] bits from the LUT control registers (LUTnCTRLB and LUTnCTRLC):

Figure 3-2. LUTn Control B Register

Bit	7	6	5	4	3	2	1	0
	INSEL1[3:0]				INSEL0[3:0]			
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Figure 3-3. LUTn Control C Register

Bit	7	6	5	4	3	2	1	0
					INSEL2[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

The table below summarizes the INSEL[3:0] options for all inputs.

Figure 3-4. CCL Input Selection Options

Value	Input source	INSEL0	INSEL1	INSEL2
0x00	MASK	None		
0x01	FEEDBACK	LUTn		
0x02	LINK	LUT(n+1)		
0x03	EVENT0	Event input source 0		
0x04	EVENT1	Event input source 0		
0x05	IO	IN0	IN1	IN2
0x06	AC	AC0 OUT		
0x07	-			
0x08	USART	USART0 TXD	USART1 TXD	USART2 TXD
0x09	SPI	SPI0 MOSI	SPI0 MOSI	SPI0 SCK
0x0A	TCA0	WO0	WO1	WO2
0x0B	-			
0x0C	TCB	TCB0 WO	TCB1 WO	TCB2 WO
Other	-			

This translates to the following code:

```
CCL.LUT1CTRLB = CCL_INSEL0_IO_gc | CCL_INSEL1_IO_gc;

CCL.LUT1CTRLC = CCL_INSEL2_IO_gc;
```

The next step is to configure the Truth tables for LUT1 to generate the right combinational logic to implement an AND gate on the selected pins. Thus, the Truth table will have a value of 0x80.

```
CCL.TRUTH1 = 0x80;
```

The next step is to configure the output of decoder, specifically, the I/O Port pin (PC3) in this example. This is done by setting the OUTEN bit on the LUT0CTRLA register.

Figure 3-5. LUTn Control A Register

Bit	7	6	5	4	3	2	1	0
	EDGEDET	OUTEN	FILTSEL[1:0]		CLKSRC[2:0]			ENABLE
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

This translates to the following code:

```
CCL.LUT1CTRLA = CCL_OUTEN_bm;
```

By enabling the LUTn output on the I/O pin, the settings for the corresponding pin are overwritten. To enable the decoding of the input sequence, the CCL and the used LUTs need to be enabled. That is done using the ENABLE bit from the LUTnCTRLA register.

```
CCL.LUT1CTRLA |= CCL_ENABLE_bm;
```

To complete the setup, the CCL module should also be enabled using a CCL global Enable bit from the CTRLA register.

Figure 3-6. CCL Control A register

Bit	7	6	5	4	3	2	1	0
		RUNSTDBY						ENABLE
Access		R/W						R/W
Reset		0						0

```
CCL.CTRLA = CCL_ENABLE_bm;
```

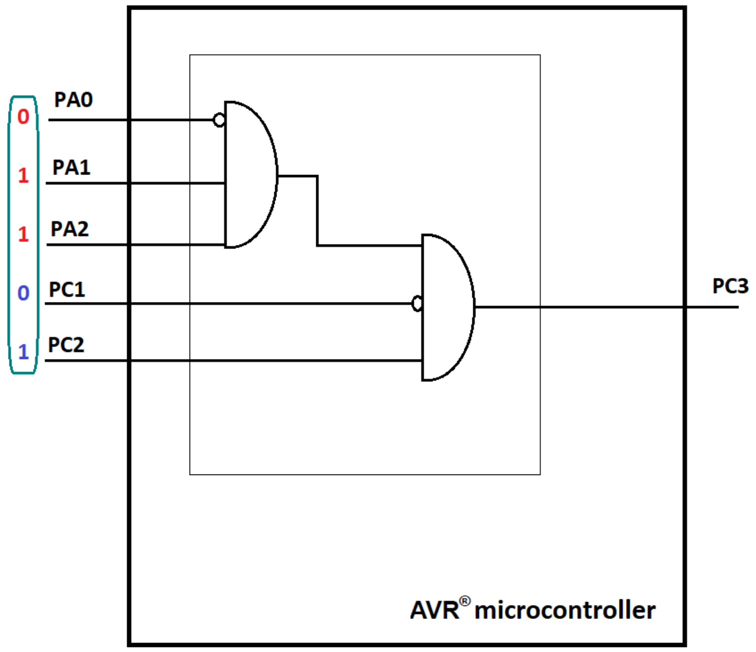
View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in the Appendix section.

4. State Decoder

The application may need to detect when a specific combination of signals (pattern) appears on the pins. By combining logic gates, a simple state decoder for external signals can be implemented without involving the CPU.

Figure 4-1. Using the AVR Microcontroller as a State Decoder



In this example, the CCL module will be used to decode the presence of the `b'10110` pattern on the input pins. LUT0 and LUT1, connected to the corresponding input pins, will be used.

The input selection from different input options is done using the INSELx[3:0] bits from the LUT Control registers (LUTnCTRLB and LUTnCTRLC), as shown in the following figures.

Figure 4-2. LUTn Control B Register

Bit	7	6	5	4	3	2	1	0
	INSEL1[3:0]				INSEL0[3:0]			
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Figure 4-3. LUTn Control C Register

Bit	7	6	5	4	3	2	1	0
					INSEL2[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

The table below summarizes the INSEL[3:0] options for all inputs.

Figure 4-4. CCL Input Selection Options

Value	Name	Description
0x0	MASK	None (masked)
0x1	FEEDBACK	Feedback input
0x2	LINK	Output from LUTn+1
0x3	EVENT0	Event input source 0
0x4	EVENT1	Event input source 1
0x5	IO	I/O-pin LUTn-IN0
0x6	AC0	AC0 out
0x7	-	Reserved
0x8	USART2	USART2 TXD
0x9	SPI0	SPI0 SCK
0xA	TCA0	TCA0 WO1
0xB	-	Reserved
0xC	TCB2	TCB2 WO
Other	-	Reserved

For this example, two adjacent LUTs (LUT0 and LUT1) will be used, with the output of LUT1 connected to the LUT0 input (linked):

```
CCL.LUT0CTRLC = CCL_INSEL2_LINK_gc;
```

The other two inputs of LUT0 and all three inputs of LUT1 are connected to the I/O pins:

```
CCL.LUT0CTRLB = CCL_INSEL0_IO_gc | CCL_INSEL1_IO_gc;
CCL.LUT1CTRLB = CCL_INSEL0_IO_gc | CCL_INSEL1_IO_gc;

CCL.LUT1CTRLC = CCL_INSEL2_IO_gc;
```

The following step is to configure the truth tables for LUT0 and LUT1 in order to generate the right combinational logic to detect `b'10110` on the selected pins. The TRUTH1 table is used to decode the pattern for the Most Significant three bits (`b'10110`).

```
CCL.TRUTH1 = 0x20;
```

LUT0 has as inputs two Least Significant bits from the input pattern (`b'10110`) and the decoded output of LUT1 on the third input, resulting in binary sequence `b'110` to be decoded. The value of the truth table in this case will be 0x40.

```
CCL.TRUTH0 = 0x40;
```

The next step is to configure the output of the decoder, specifically, the I/O PORT pin PA3 in this example. This is done by setting the OUTEN bit on the LUT0CTRLA register.

Figure 4-5. LUTn Control A Register

Bit	7	6	5	4	3	2	1	0
	EDGEDET	OUTEN	FILTSEL[1:0]		CLKSRC[2:0]			ENABLE
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

This translates to the following code:

```
CCL.LUT0CTRLA = CCL_OUTEN_bm;
```

By enabling the LUTn output on the I/O pin, the settings for the corresponding pin are overwritten. To complete the setup and the start decoding of the input sequence, the CCL and used LUTs need to be enabled. That is done using the ENABLE bit from the LUTnCTRLA register.

```
CCL.LUT1CTRLA = CCL_ENABLE_bm;
CCL.LUT0CTRLA |= CCL_ENABLE_bm;
```

To complete the setup, the CCL module should also be enabled using a CCL global Enable bit from the CTRLA register.

Figure 4-6. CCL Control A register

Bit	7	6	5	4	3	2	1	0
		RUNSTDBY						ENABLE
Access		R/W						R/W
Reset		0						0

```
CCL.CTRLA = CCL_ENABLE_bm;
```

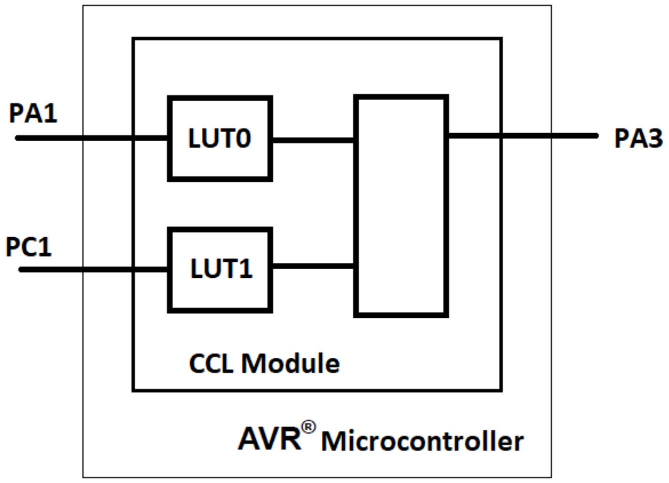
View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in the Appendix section.

5. SR Latch

This chapter describes an application example that uses CCL combinational and sequential logic to implement a SR latch. This functionality can be created using two adjacent LUTs (LUT0 and LUT1) connected through a sequential logic block.

Figure 5-1. Using CCL to Implement an SR Latch



For Set and Reset signals, two pins are used as inputs for LUTs (I/O PORT pin PA1 and I/O PORT pin PC1). That translates to following code:

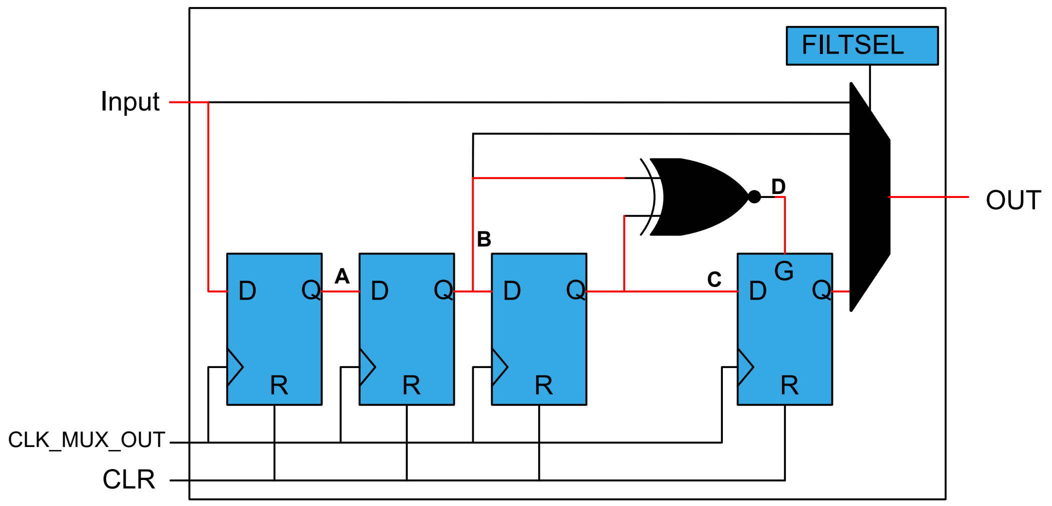
```
CCL.LUT0CTRLB = CCL_INSEL0_MASK_gc | CCL_INSEL1_IO_gc;
CCL.LUT0CTRLC = CCL_INSEL2_MASK_gc;
CCL.LUT1CTRLB = CCL_INSEL0_MASK_gc | CCL_INSEL1_IO_gc;
CCL.LUT0CTRLC = CCL_INSEL2_MASK_gc;
```

In this case, only the input selected for the Input signal needs to be considered when configuring the Truth register for each LUT. For instance, if the signal is active-high and available on LUTn_IN[1], the Truth register should be set to 0x02. If the input signal is active-low, which is the case for many evaluation kits, the Truth register should be set to 0x00. For the selected example, the input signals are active-low, so the Truth register will be set to 0x00 for both LUTs:

```
CCL.TRUTH0 = 0x00;
CCL.TRUTH1 = 0x00;
```

The truth table output is a combinatorial function of the inputs. This may cause some short glitches when the inputs change value. These glitches may not cause any problems, but if the LUT output is set to trigger an event, used as input on a timer or similar, an unwanted glitch may trigger unwanted events and peripheral action. In removing these glitches by clocking through the filters, the user will only get the intended output. Each Look-Up Table (LUT) in the CCL includes a filter that can be used to synchronize or filter the LUT output.

Figure 5-2. CCL Filter



The selection of the filter option is done by using the FILTSEL[1:0] bits from the LUTnCTRLA register.

Figure 5-3. CCL Filter Options

Value	Name	Description
0x0	DISABLE	Filter disabled
0x1	SYNCH	Synchronizer enabled
0x2	FILTER	Filter enabled
0x3	-	Reserved

```
CCL.LUT0CTRLA = CCL_FILTSEL_FILTER_gc;
CCL.LUT1CTRLA = CCL_FILTSEL_FILTER_gc;
```

The next step is to connect the LUTs through a sequential logic to create SR latch functionality. The bits in SEQSEL0[3:0] from the Sequential Control register (SEQCTRL0) select the sequential configuration for LUT0 and LUT1.

Figure 5-4. Sequential 1 Control 0 Register

Bit	7	6	5	4	3	2	1	0
					SEQSELn[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:0 – SEQSELn[3:0]: Sequential Selection bits
The bits in SEQSELn select the sequential configuration for LUT[2n] and LUT[2n+1].

Value	Name	Description
0x0	DISABLE	Sequential logic is disabled
0x1	DFF	D flip flop
0x2	JK	JK flip flop
0x3	LATCH	D latch
0x4	RS	RS latch
Other	-	Reserved

This translates to the following code:

```
CCL.SEQCTRL0 = CCL_SEQSEL0_RS_gc;
```

To complete the setup and enable the LUT0 output on the LUT0OUT pin (PA3), the used LUTs and CCL need to be enabled.

```
CCL.LUT1CTRLA |= CCL_ENABLE_bm;
CCL.LUT0CTRLA |= CCL_ENABLE_bm | CCL_OUTEN_bm;
CCL.CTRLA = CCL_ENABLE_bm;
```

View Code Example on GitHub
Click to browse repository

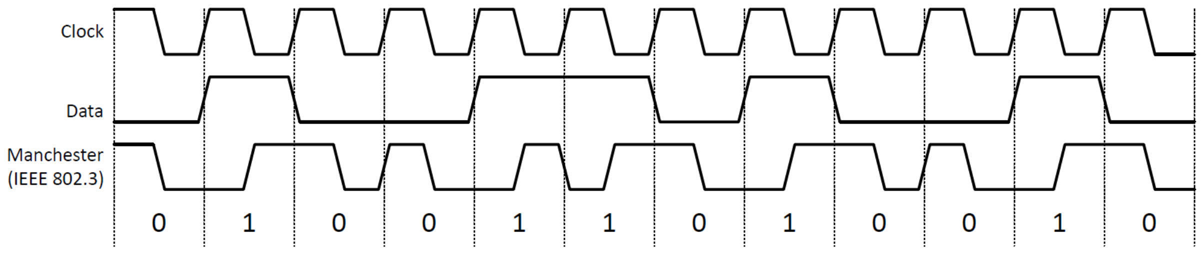
Tip: The full code example is also available in the Appendix section.

6. Manchester Encoder

In this section, CCL and SPI will be used to build a Manchester encoder and to transmit a Manchester-encoded signal. This requires very few CPU cycles to load the input data to SPI, and the rest of the work is performed by the CCL and SPI modules.

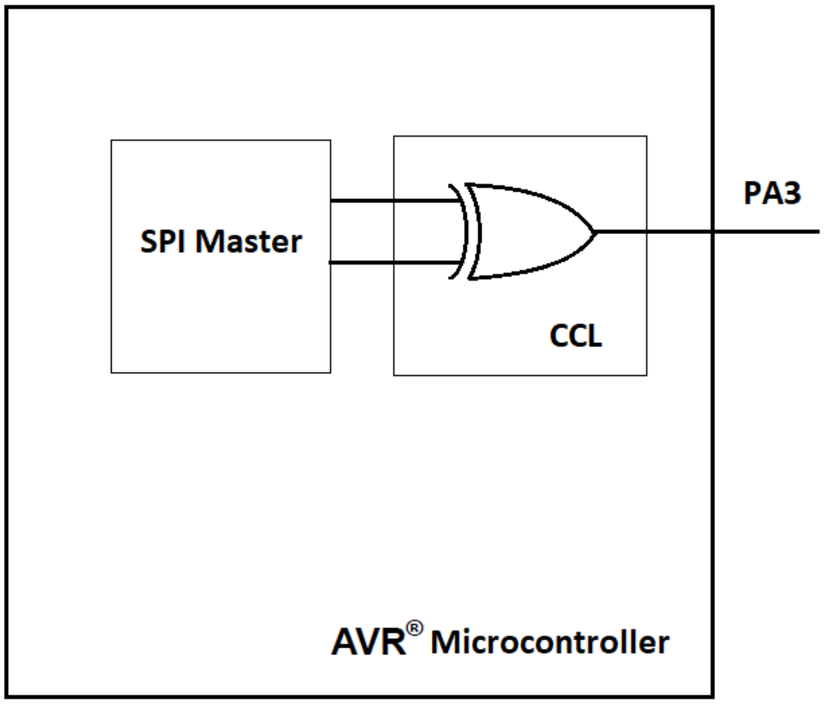
The Manchester code is a type of line code that has constant DC component. There are two versions of the Manchester code; this document refers to the version defined in the IEEE 802.3 standard. The Manchester code combines data and clock into a single signal, where one clock cycle is a Manchester-bit period. A transition always occurs in the middle of the bit period. DATA = `0` is represented by a falling edge (high-to-low transition) in the middle of the bit period, and DATA = `1` is represented by a rising edge (low-to-high transition) in the middle of the bit period. An example is shown below.

Figure 6-1. Manchester Encoder Data



One way to obtain the Manchester encoded data is to use the XOR function between clock and data. In the current example, the SPI module is used to generate clock and data signals for the encoder.

Figure 6-2. Using CCL and SPI as Manchester Encoder



The SPI module should be configured as master to generate signals on the outputs.

```
void SPI0_init()
{

    SPI0.CTRLA = SPI_ENABLE_bm
                 | SPI_MASTER_bm;
}
```

LUT0 is then configured to use SPI MOSI and SPI SCK signals as inputs. The unused LUT0 input will be masked.

```
CCL.LUT0CTRLB = CCL_INSEL0_MASK_gc | CCL_INSEL1_SPI0_gc;

CCL.LUT0CTRLC = CCL_INSEL2_SPI0_gc;
```

To create a logic XOR function between LUT0_IN[1] and LUT0_IN[2], the value of the truth table should be 0x14.

```
CCL.TRUTH0 = 0x14;
```

To complete the setup and enable the LUT0 output on the LUT0OUT pin (PA3), CCL and LUT0 need to be enabled.

```
CCL.LUT0CTRLA = CCL_ENABLE_bm | CCL_OUTEN_bm;

CCL.CTRLA = CCL_ENABLE_bm;
```

View Code Example on GitHub
Click to browse repository

Tip: The full code example is also available in the Appendix section.

7. References

1. ATmega4809 product page: https://www.microchip.com/wwwproducts/en/ATMEGA4809
2. megaAVR® 0-Series Manual (DS40002015)
3. Getting Started with Core Independent Peripherals on AVR® (DS00002451)
4. ATmega4809 Xplained Pro web page: https://www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro.

8. Appendix

Example 8-1. Logic AND Gate Code Example

```
#include <avr/io.h>

void PORT0_init (void);
void CCL0_init(void);

/**
 * \brief Initialize ports
 */
void PORT0_init (void)
{
    PORTC.DIR &= ~PIN0_bm;          //PC0 - LUT1 IN[0]
    PORTC.DIR &= ~PIN1_bm;          //PC1 - LUT1 IN[1]
    PORTC.DIR &= ~PIN2_bm;          //PC2 - LUT1 IN[2]

    PORTC.DIR |= PIN3_bm;            //PC3 - LUT1 output
}

/**
 * \brief Initialize CCL peripheral
 */
void CCL0_init(void)
{

//configure inputs for used LUTs
    CCL.LUT1CTRLB = CCL_INSEL0_IO_gc    /* IO pin LUTn-IN0 input source */
                  | CCL_INSEL1_IO_gc; /* IO pin LUTn-IN1 input source */
    CCL.LUT1CTRLC = CCL_INSEL2_IO_gc;   /* IO pin LUTn-IN2 input source */

//Configure Truth Table
    CCL.TRUTH1 = 0x80; /* Truth 1: 128 */

//Enable LUT0 output on IO pin
    CCL.LUT1CTRLA = CCL_OUTEN_bm;     /* Output Enable: enabled */

//Enable LUTs
    CCL.LUT1CTRLA |= CCL_ENABLE_bm;    /* LUT Enable: enabled */

//Enable CCL module
    CCL.CTRLA = CCL_ENABLE_bm;         /* Enable: enabled */
}


int main(void)
{
    PORT0_init();
    CCL0_init();
    while (1)
    {
        ;
    }
}
```

Example 8-2. State Decoder Code Example

```
#include <avr/io.h>

void PORT0_init (void);
void CCL0_init(void);

/**
 * \brief Initialize ports
 */
void PORT0_init (void)
{
    PORTA.DIR &= ~PIN0_bm;          //PA0 - LUT0 IN[0]
    PORTA.DIR &= ~PIN1_bm;          //PA1 - LUT0 IN[1]
    PORTC.DIR &= ~PIN0_bm;          //PC0 - LUT1 IN[0]
    PORTC.DIR &= ~PIN1_bm;          //PC0 - LUT1 IN[1]
    PORTC.DIR &= ~PIN2_bm;          //PC0 - LUT1 IN[2]

    PORTA.DIR |= PIN3_bm;            //PA3 - LUT0 output

}

/**
 * \brief Initialize CCL peripheral
 */
void CCL0_init(void)
{

//configure inputs for used LUTs
    CCL.LUT0CTRLB = CCL_INSEL0_IO_gc    /* IO pin LUTn-IN0 input source */
                  | CCL_INSEL1_IO_gc; /* IO pin LUTn-IN1 input source */
    CCL.LUT0CTRLC = CCL_INSEL2_LINK_gc; /* Linked LUT input source */

    CCL.LUT1CTRLB = CCL_INSEL0_IO_gc    /* IO pin LUTn-IN0 input source */
                  | CCL_INSEL1_IO_gc; /* IO pin LUTn-IN1 input source */
    CCL.LUT1CTRLC = CCL_INSEL2_IO_gc;   /* IO pin LUTn-IN2 input source */

//Configure Truth Tables
    CCL.TRUTH0 = 0x40; /* Truth 0: 64 */
    CCL.TRUTH1 = 0x20; /* Truth 1: 32 */

//Enable LUT0 output on IO pin
    CCL.LUT0CTRLA = CCL_OUTEN_bm;     /* Output Enable: enabled */

//Enable LUTs
    CCL.LUT0CTRLA |= CCL_ENABLE_bm;    /* LUT Enable: enabled */
    CCL.LUT1CTRLA = CCL_ENABLE_bm;    /* LUT Enable: enabled */

//Enable CCL module
    CCL.CTRLA = CCL_ENABLE_bm;         /* Enable: enabled */
}

int main(void)
{
    PORT0_init();
    CCL0_init();
    while (1)
    {
        ;
    }
}
```

Example 8-3. SR Latch Code Example

```
#include <avr/io.h>

void PORT0_init (void);
void CCL0_init(void);

/**
 * \brief Initialize ports
 */
void PORT0_init (void)
{

    PORTA.DIR &= ~PIN1_bm;           //PA1 - LUT0 IN[1]
    PORTC.DIR &= ~PIN1_bm;           //PC1 - LUT1 IN[1]

    PORTA.DIR |= PIN3_bm;            //PA3 - LUT0 output

}

/**
 * \brief Initialize CCL peripheral
 */
```

```
void CCL0_init(void)
{

//configure inputs for used LUTs
    CCL.LUT0CTRLB = CCL_INSEL0_MASK_gc    /* LUTn-IN0 input masked */
                  | CCL_INSEL1_IO_gc; /* IO pin LUTn-IN1 input source */
    CCL.LUT0CTRLC = CCL_INSEL2_MASK_gc; /* LUTn-IN2 input masked */

    CCL.LUT1CTRLB = CCL_INSEL0_MASK_gc    /* LUTn-IN0 input masked */
                  | CCL_INSEL1_IO_gc; /* IO pin LUTn-IN1 input source */
    CCL.LUT1CTRLC = CCL_INSEL2_MASK_gc; /* LUTn-IN2 input masked */

//Configure Truth Tables
    CCL.TRUTH0 = 0x00; /* Truth 0: 0 */
    CCL.TRUTH1 = 0x00; /* Truth 0: 0 */

// Configure filter
    CCL.LUT0CTRLA = CCL_FILTSEL_FILTER_gc;      /* Enable filter*/
    CCL.LUT1CTRLA = CCL_FILTSEL_FILTER_gc;      /* Enable filter*/

//Enable sequential logic for LUT0 and LUT1
    CCL.SEQCTRL0 = CCL_SEQSEL0_RS_gc;

//Enable LUT0 output on IO pin
    CCL.LUT0CTRLA |= CCL_OUTEN_bm;     /* Output Enable: enabled */

//Enable LUTs
    CCL.LUT0CTRLA |= CCL_ENABLE_bm;    /* LUT Enable: enabled */
    CCL.LUT1CTRLA |= CCL_ENABLE_bm;    /* LUT Enable: enabled */

//Enable CCL module
    CCL.CTRLA = CCL_ENABLE_bm;         /* Enable: enabled */
}


int main(void)
{
    PORT0_init();
    CCL0_init();
    /* Replace with your application code */
    while (1)
    {
        ;
    }
}
```

Example 8-4. Manchester Encoder Code Example

```
#include <avr/io.h>

void PORT0_init (void);
void CCL0_init(void);

void SPI0_init(void);
void slaveSelect(void);
void slaveDeselect(void);
uint8_t SPI0_exchangeData(uint8_t data);

void SPI0_init(void)
{
    SPI0.CTRLA = SPI_CLK2X_bm            /* Enable double-speed */
               | SPI_DORD_bm             /* LSB is transmitted first */
               | SPI_ENABLE_bm           /* Enable module */
               | SPI_MASTER_bm           /* SPI module in Master mode */
               | SPI_PRESC_DIV16_gc;     /* System Clock divided by 16 */
}

uint8_t SPI0_exchangeData(uint8_t data)
{
    SPI0.DATA = data;

    while (!(SPI0.INTFLAGS & SPI_IF_bm))  /* waits until data is exchanged*/
    {
        ;
    }
    return SPI0.DATA;
}


/**
 * \brief Initialize ports
 */
void PORT0_init (void)
{

    PORTA.DIR |= PIN4_bm;     /* Set MOSI pin direction to output */
    PORTA.DIR &= ~PIN5_bm;     /* Set MISO pin direction to input */
    PORTA.DIR |= PIN6_bm;     /* Set SCK pin direction to output */

    PORTA.DIR |= PIN3_bm;     /* Set PA3 as LUT0 output */

}

/**
 * \brief Initialize CCL peripheral
 */
void CCL0_init(void)
{

//configure inputs for used LUTs
    CCL.LUT0CTRLB = CCL_INSEL0_MASK_gc    /* Masked input */
                    | CCL_INSEL1_SPI0_gc; /* SPI0 MOSI input source */
    CCL.LUT0CTRLC = CCL_INSEL2_SPI0_gc;   /* SPI0 SCK source */

//Configure Truth Tables
    CCL.TRUTH0 = 0x14; /* Truth 0: 20 */

//Enable LUT0 output on IO pin
    CCL.LUT0CTRLA = CCL_OUTEN_bm;     /* Output Enable: enabled */

//Enable LUTs
    CCL.LUT0CTRLA |= CCL_ENABLE_bm;    /* LUT Enable: enabled */

//Enable CCL module
    CCL.CTRLA = CCL_ENABLE_bm;         /* Enable: enabled */
}


int main(void)
{
    PORT0_init();
    SPI0_init();
    CCL0_init();
    while (1)
    {
        SPI0_exchangeData(0xA5);
    }
}
```

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.

Legal Notice

Trademarks

© 2018, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-4004-8

Quality Management System Certified by DNV

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-72884388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820