AN2451

Getting Started with Core Independent Peripherals on AVR®

Features

- Introduction to Configurable Custom Logic (CCL)
- Introduction to Event System (EVSYS)
- Core Independent Application Example
  - Connecting peripherals through the Event System
  - Filtering button signal using the CCL and alternative clock signal
  - Triggering ADC conversion from filtered button signal

Introduction

Core Independent Peripherals (CIPs) is a category of peripherals available on many AVR® devices. This application note focuses on the tinyAVR® 1-Series, but the general principles apply across all devices equipped with CIPs, even though the specific peripheral features and design may vary.

A CIP is designed to handle its tasks among one or multiple peripherals with no code or supervision from the CPU to maintain the operation. This brings up many advantages, such as providing short and predictable response times between peripherals, reducing the complexity and execution time of the software, as well as the possibility of reduced power consumption.

There is a number of CIPs available on devices in the tinyAVR® 1-Series. Examples are: Event System (EVSYS), Configurable Custom Logic (CCL), Timer/Counter A and B (TCA/TCB), Real Timer Counter (RTC), Analog-to-Digital Converter (ADC), and CRCSCAN.

This application note will first introduce the two most powerful building blocks in a core independent application: the CCL and the Event System. Then, an application example that combines the CCL, Event System, RTC, and ADC to filter the signal from a button and initiate an ADC conversion core independently, is presented. This should help users start building their own projects using CIPs.

Table of Contents

1. Relevant Devices

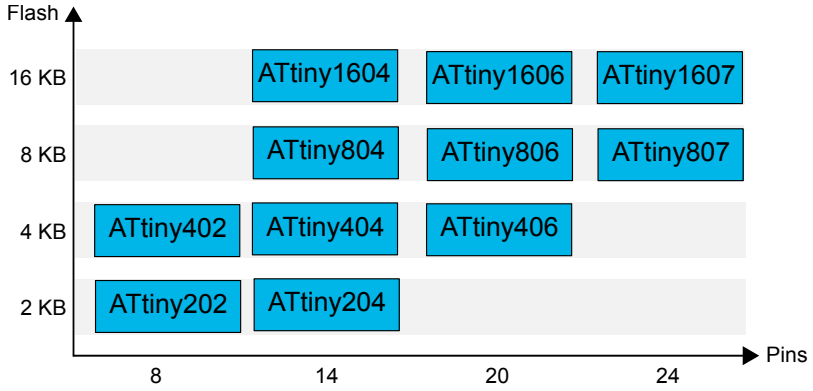
This chapter lists the relevant devices for this document.

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



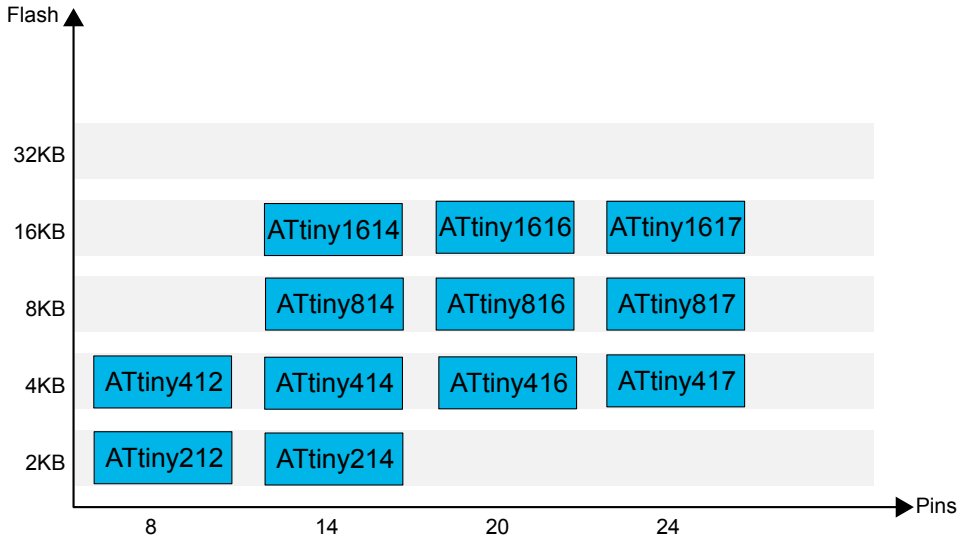
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.2 tinyAVR® 1-Series

The figure below shows the tinyAVR® 1-series devices, illustrating 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 Industrial Device Overview



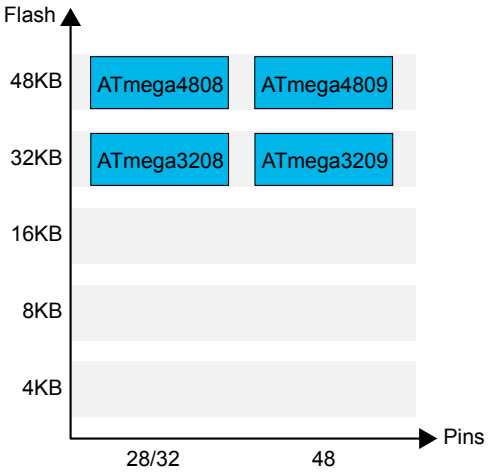
Devices with different Flash memory size typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-Series

Figure 1-3 shows the feature compatible devices in the megaAVR® device family, including pinout variants and memory variants.

- 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. Device Family Overview



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

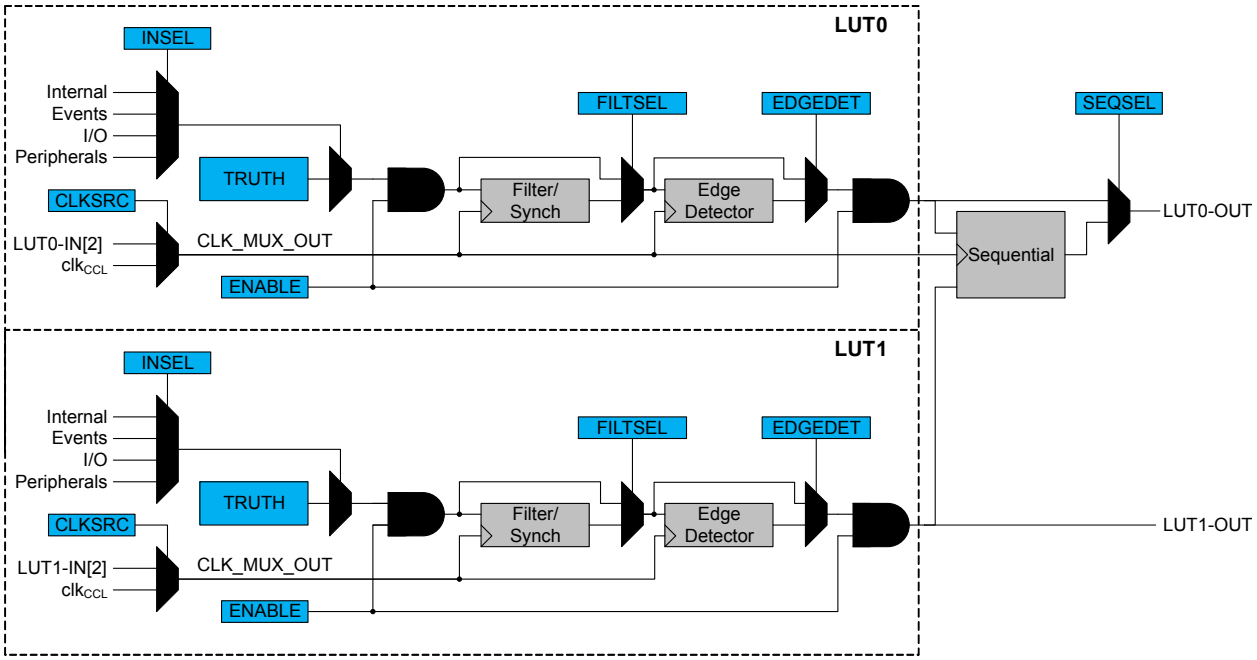
2. Introduction to CCL

The Configurable Custom Logic (CCL) is a programmable logic peripheral, which can be connected to a wide range of internal and external inputs such as device pins, events, or other internal peripherals. The CCL can serve as "glue logic" between the device peripherals and external devices.

The CCL peripheral has one pair of LookUp Tables (LUT). Each LUT consists of three inputs, 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 and any device that have CCL will have a minimum of two LUTs available. Inputs can be individually masked. The output can be generated from the inputs combinatorialy, 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 (LUT0/LUT1) outputs, enabling complex waveform generation.

Using the CCL can eliminate the need for additional external logic components and provide the core with support to handle time critical parts of the application.

Figure 2-1. CCL Overview



2.1 Truth Table

By using the look-up table in the LUT it is possible to generate any logical expression with up to three inputs.

The inputs can be individually:

- Masked
- Connected to I/Os
- Driven by peripherals:
  - Analog comparator output (AC)
  - Timer/Counters waveform outputs (TC)
  - USART

  - SPI
- Driven by internal events from the Event System
- Driven by other CCL sub-modules

Understanding how to use the truth table to generate the logical expression needed is the key to make the CCL work as intended.

Each TRUTH[x] line in the table will create one 3-input gate, and by choosing more than one TRUTH in the table it is possible to create complex logical expressions. Each combination of the input bits (IN[2:0]) corresponds to one bit in the TRUTHn register.

Table 2-1. LUT Truth Table

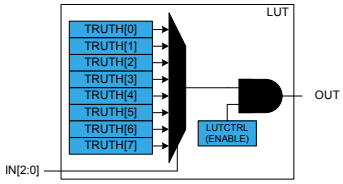
IN[2]	IN[1]	IN[0]	OUT
0	0	0	TRUTH[0]
0	0	1	TRUTH[1]
0	1	0	TRUTH[2]
0	1	1	TRUTH[3]
1	0	0	TRUTH[4]
1	0	1	TRUTH[5]
1	1	0	TRUTH[6]
1	1	1	TRUTH[7]

Table 2-2. Possible Logic Blocks

IN[]	TRUTH	AND	NAND	OR	NOR	XOR	XNOR	NOT
000	TRUTH[0]	0	1	0	1	0	1	1
001	TRUTH[1]	0	1	1	0	1	0	x
010	TRUTH[2]	0	1	1	0	1	0	x
011	TRUTH[3]	0	1	1	0	0	1	x
100	TRUTH[4]	0	1	1	0	1	0	x
101	TRUTH[5]	0	1	1	0	0	1	x
110	TRUTH[6]	0	1	1	0	0	1	x
111	TRUTH[7]	1	0	1	0	1	0	0
		0x80	0x7F	0xFE	0x01	0x96	0x69	0x01

Each TRUTH[x] chosen will be OR-ed together creating the final logical expression.

Figure 2-2.



2.1.1 Creating Simple Logic Blocks

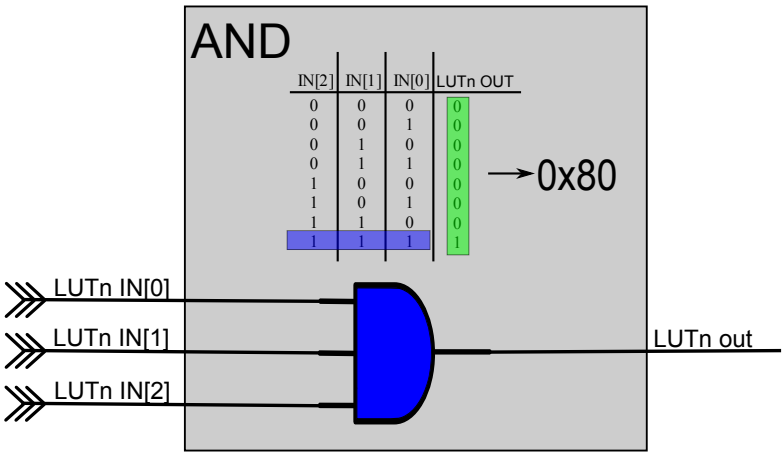
On each of the LUTs, it is possible to create simple logical blocks as AND, OR, NAND, NOR, and XOR using the truth table with up to three inputs.

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

2.1.1.1 AND Gate

To get a HIGH(1) output from an AND gate, all inputs must be HIGH(1). Looking at the truth table, 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), giving the hex value 0x80 to put into the TRUTHn register.

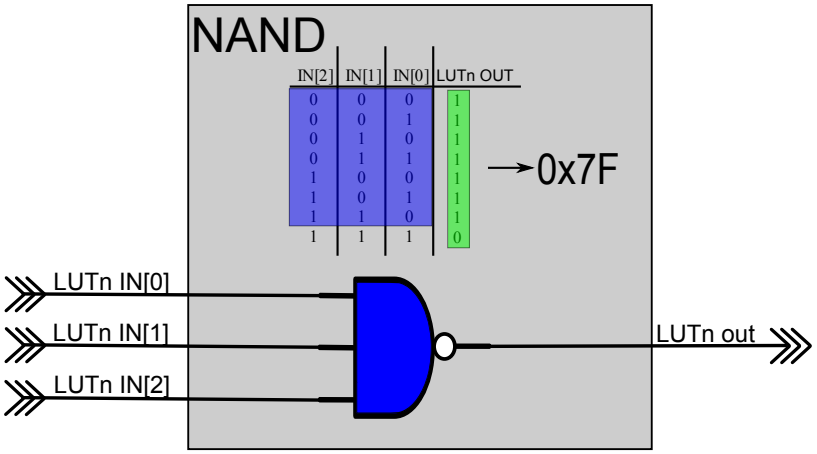
Figure 2-3. AND Gate



2.1.1.2 NAND Gate

To get a HIGH(1) output from a NAND gate, one or more of the inputs must be LOW(0). If all inputs are HIGH(1) the output will be LOW(0). Looking at the truth table, all except TRUTH[7] fulfill this requirement. This means that TRUTH[0] to TRUTH[6] must be high and TRUTH[7] must be low, giving the hex value 0x7F to put into the TRUTHn register.

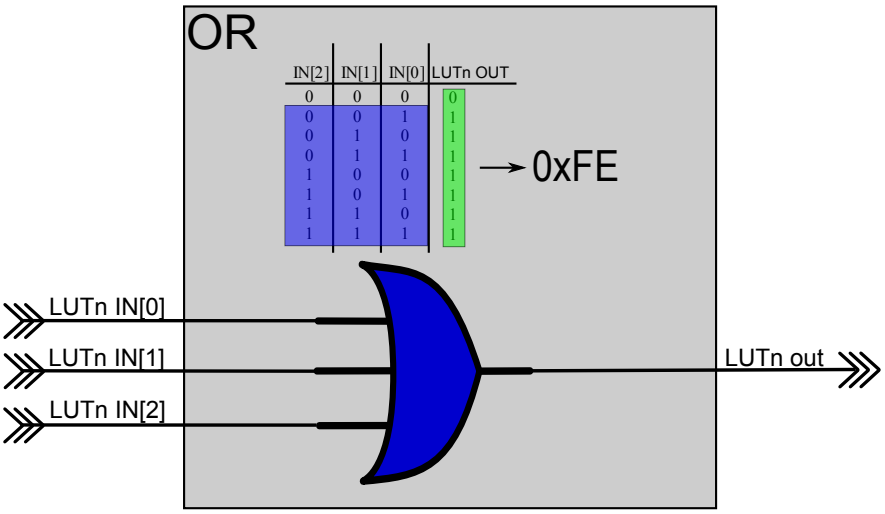
Figure 2-4. NAND Gate



2.1.1.3 OR Gate

To get a HIGH(1) output from an OR gate, one or more of the inputs must be HIGH(1). If all inputs are LOW(0) the output will be LOW(0). Looking at the truth table, all except TRUTH[0] fulfill this requirement. This means that TRUTH[1] to TRUTH[7] must be HIGH(1) and TRUTH[0] must be LOW(0), giving the hex value 0xFE to put into the TRUTHn register.

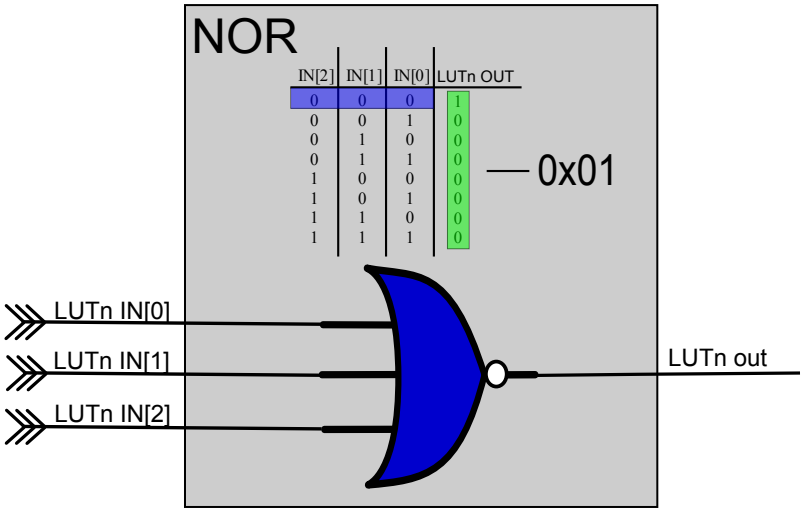
Figure 2-5. OR Gate



2.1.1.4 NOR Gate

To get a HIGH(1) output from a NOR gate, all the inputs must be LOW(0). If any of the inputs are HIGH(1) the output will be LOW(0). Looking at the truth table, only TRUTH[0] fulfill this requirement. This means that TRUTH[1] to TRUTH[7] must be LOW(0) and TRUTH[0] must be HIGH(1), giving the hex value 0x01 to put into the TRUTHn register.

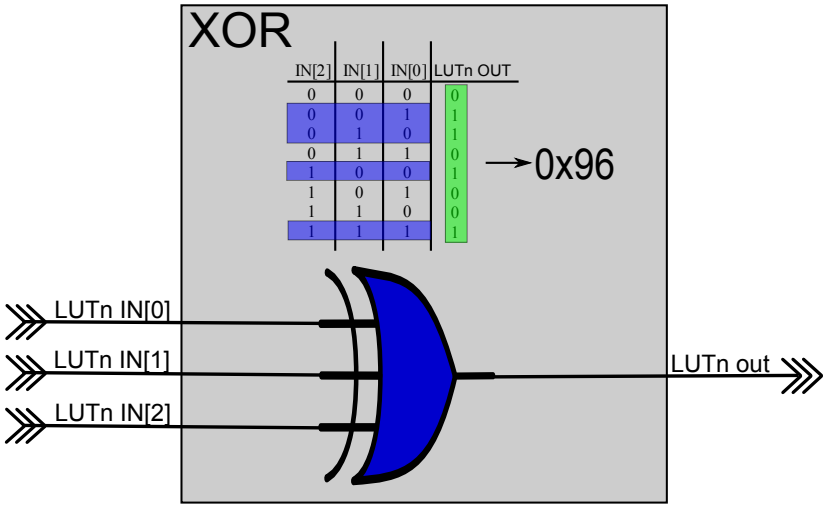
Figure 2-6. NOR Gate



2.1.1.5 XOR Gate

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

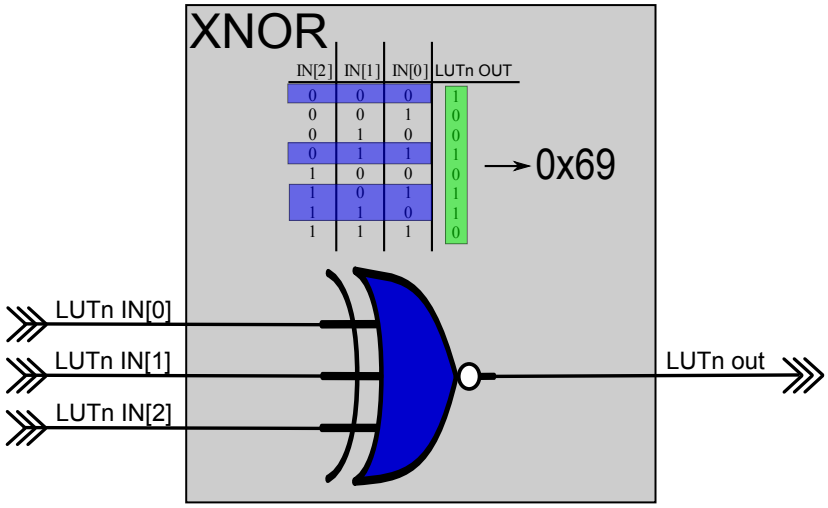
Figure 2-7. XOR Gate



2.1.1.6 XNOR Gate

To get a HIGH(1) output from an XNOR gate, the number of LOW(0) inputs must be odd. Looking at the truth table, TRUTH[0], TRUTH[3],TRUTH[5], and TRUTH[6] fulfill this requirement. This means that these must be HIGH(1) and the rest must be LOW(0), giving the hex value 0x69 to put into the TRUTHn register.

Figure 2-8. XNOR Gate



2.1.2 Masking Inputs

Each LUT have three inputs that can be used. When not all the three inputs are needed, the unused input can be masked (tied low). Only the TRUTH bits were the masked input is '0' is can be used when looking at the truth table to determine how the bits should be set to get the wanted logic.

When masking one input, the truth table can be simplified to have only two inputs and when masking two inputs it can be reduced to have only one input

The table below shows an example of the truth table when masking IN[0].

Table 2-3. LUT Truth Table when IN[0] is Masked

IN[2]	IN[1]	OUT
0	0	TRUTH[0]
0	1	TRUTH[2]
1	0	TRUTH[4]
1	1	TRUTH[6]

The table below shows an example of the truth table when masking IN[1].

Table 2-4. LUT Truth Table when IN[1] is Masked

IN[2]	IN[0]	OUT
0	0	TRUTH[0]
0	1	TRUTH[1]
1	0	TRUTH[4]
1	1	TRUTH[5]

The table below shows an example of the truth table when masking IN[2].

Table 2-5. LUT Truth Table when IN[2] is Masked

IN[1]	IN[0]	OUT
0	0	TRUTH[0]
0	1	TRUTH[1]
1	0	TRUTH[2]
1	1	TRUTH[3]

The table below shows an example of the truth table when masking IN[0] and IN[1].

Table 2-6. LUT Truth Table when IN[0] and IN[1] are Masked

IN[2]	OUT
0	TRUTH[0]
1	TRUTH[4]

The table below shows an example of the truth table when masking IN[0] and IN[2].

Table 2-7. LUT Truth Table when IN[0] and IN[2] are Masked

IN[1]	OUT
0	TRUTH[0]
1	TRUTH[2]

The table below shows an example of the truth table when masking IN[1] and IN[2].

Table 2-8. LUT Truth Table when IN[1] and IN[2] are Masked

IN[0]	OUT
0	TRUTH[0]
1	TRUTH[1]

Below are some examples of were various inputs are masked.

Figure 2-9. Two Input AND Gates, IN[0] Masked

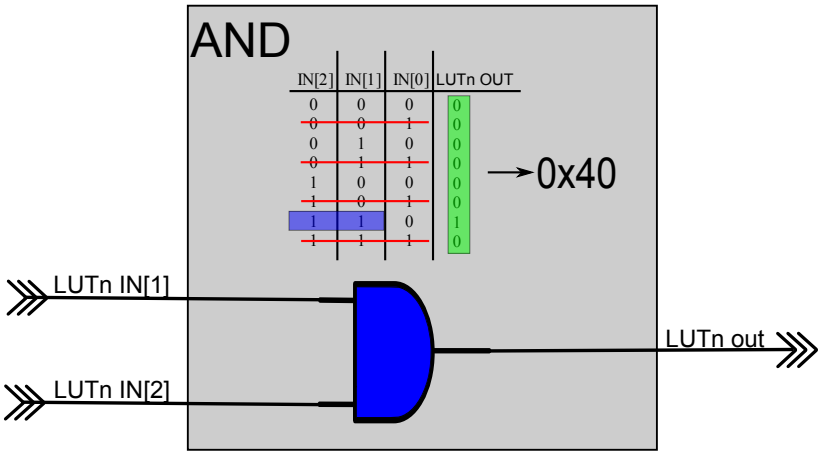


Figure 2-10. Two Input OR Gates, IN[1] Masked

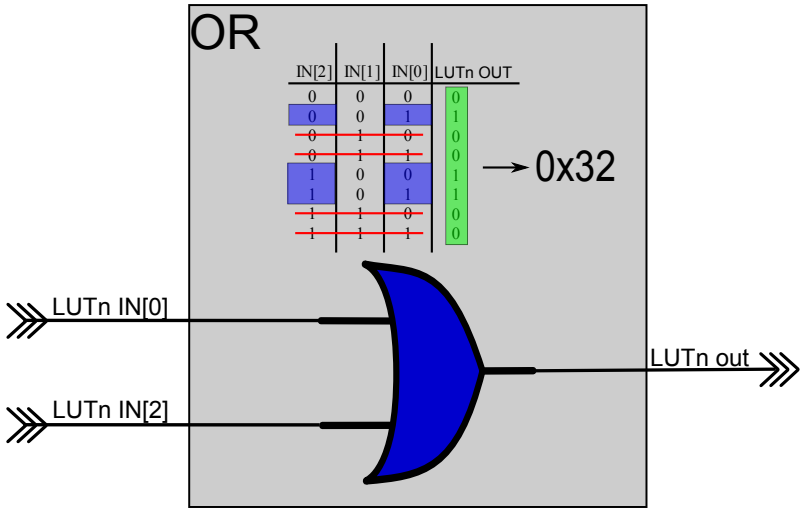
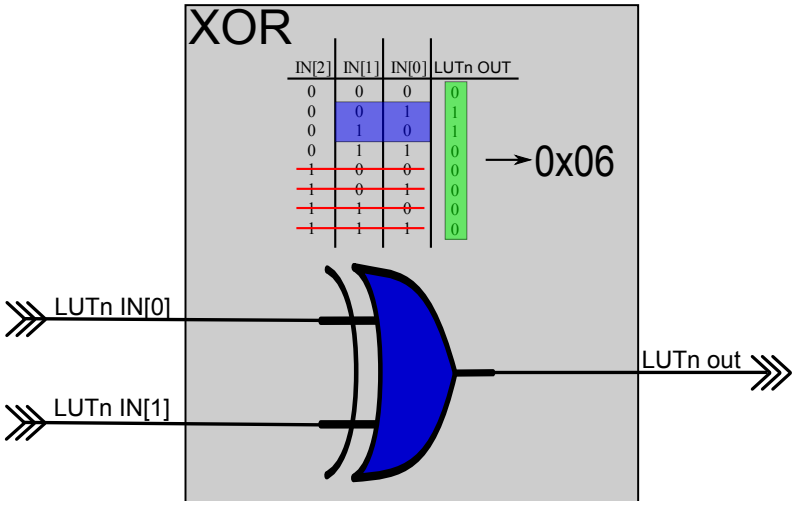


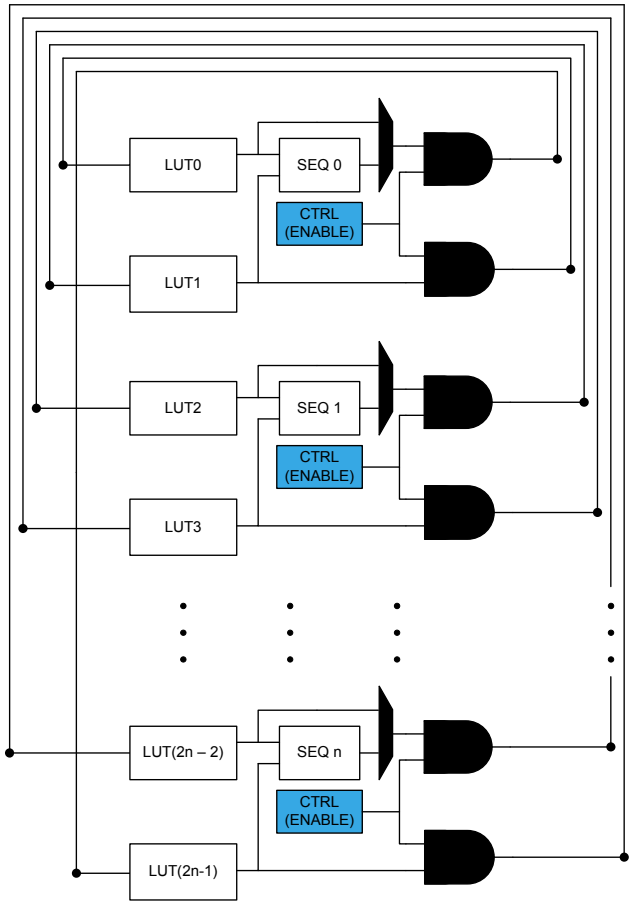
Figure 2-11. Two Input XOR Gates, IN[2] Masked



2.1.3 Linking LUTs

Linking LUTs means taking the output of one LUT and using it as an input on another LUT. Doing this it is possible to solve logical expressions with up to five inputs using two LUTs. LUTn can only LINK to LUTn +1 and the last LUT can LINK to the first LUT. The LUT output of LUTn can be linked to any of the inputs of the other LUTn+1. When creating the truth table to determine what needs to be written in the TRUTH register for each LUT, the truth tables for both LUTs should be done as if the LUTs were not linked.

Figure 2-12. Linking LUTs



In Atmel Start it is possible to find application notes and code examples using CCL and linked LUTs.

- Quadrature Decoding using CCL with TCA and TCB

2.1.4 How to Realize Logical Expressions

Using the truth table to create simple logical gates can solve a lot of tasks, but often a more complex and specific logical function is needed. Below are some examples on how logical expressions can be realized based on logical expressions with up to three inputs and how these can be solved by using one LUT, and also example on how linking two LUTs together can solve logical expression using up to five inputs.

2.1.4.1 Realize Logical Expression using One LUT

Imagine the following logical expression: $(A \bullet B) \oplus (B \bullet C)$.

This gives this truth table:

Table 2-9. LUT Truth Table

C	B	A	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0

C	B	A	OUT
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Looking at the truth table above, the value needed to be written to the TRUTH register will be 0x72.

2.1.4.2 Realize Logical Expression using Linked LUTs

Below is an example on linking LUT0 to LUT1 and how to fill out the truth tables for both LUTs.

Imagine the following logical expression: $(A \bullet B \oplus C) + (D \bullet \bar{E})$.

The truth table for LUT0 should be created. LUT0 will take care of the first part of the logical expression $(A \bullet B \oplus C)$.

This will give this truth table:

Table 2-10. LUT0 Truth Table

C	B	A	OUT
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

From the above truth table, 0x60 should go into LUT0 TRUTH register to realize the first part of the logical expression.

Now the truth table for LUT1 must be created. Before this can be done, it must be decided what input to use on LUT1. All inputs can be used and in this example LUT0 out is linked to LUT1 input 1. This will be equal to the second column of the truth table. If input 0 was used, it would have been the first column that should be used and the third if input 2 was used.

To make the development of the truth table for LUT1 easier, the expression could be simplified since LUT0 already has handled the first part. The expression can be viewed like this when creating the truth table for LUT1: $(X + D \bullet \bar{E})$ were X = $(A \bullet B \oplus C)$

Table 2-11. LUT1 Truth Table

E	X	D	OUT
0	0	0	0
0	0	1	1

E	X	D	OUT
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

From the above truth table 0xCE should go into LUT1 TRUTH register to realize the second part of the logical expression.

2.2 Two-stage Synchronizer, Filter, and Edge Detector

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. Removing these glitches by clocking through filters, the user will only get the intended output.

2.2.1 Two-stage Synchronizer

In the synchronizer option, the output signal from the truth table is clocked through a two-stage synchronizer, and the signal will be delayed up to two clock cycles when using this option. A glitch from the LUT shorter than 1 clock cycle will be filtered out using the synchronizer as long as the glitch is not present on the rising edge of the clock. Although useful in many situations, the two-stage synchronizer has limitations. If the glitch is present on the rising edge on the first stage of the synchronizer, it will latch and the glitch will become 1 clock cycle long when exiting the synchronizer.

Figure 2-13. Two-stage Synchronizer

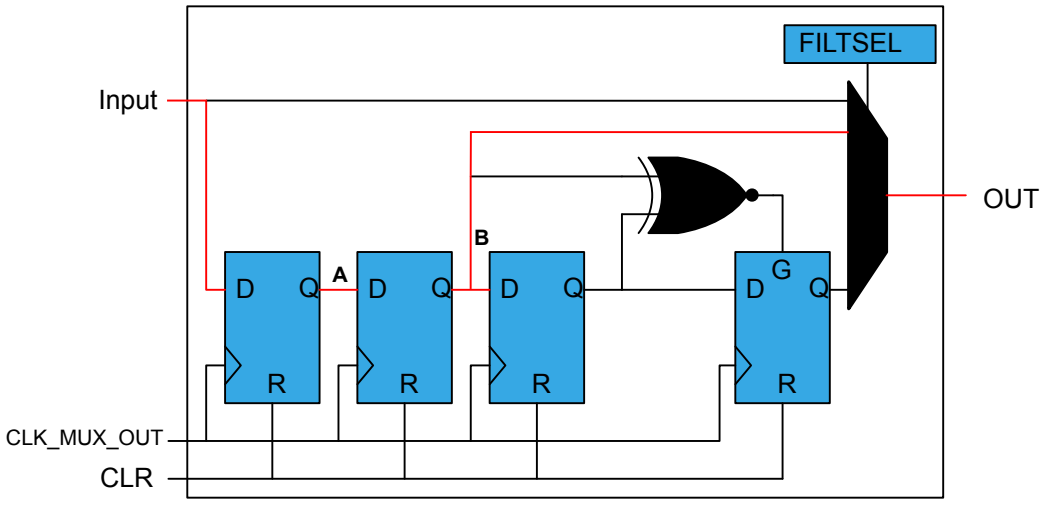
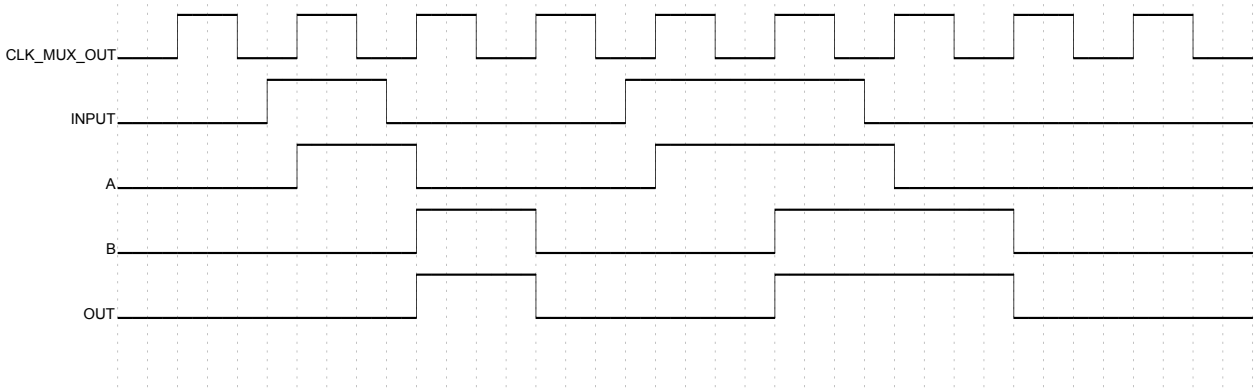


Figure 2-14. Two-stage Synchronizer Timing



2.2.2 Filter

To be sure to remove all glitches, the filter options should be selected if the user wants to avoid spikes and glitches to affect the system. The filter will first run the signal through the two-stage synchronizer and then further through the filter.

The XNOR acts as a majority vote and as long as the inputs to the XNOR are different to each other, the output will be "0".

- If the two XNOR inputs are equal its output is 1
- If the XNOR output is 1, the gate input on the last D flip-flop is high
- If the XNOR output is 0, the gate input on the last D flip-flop is low

When a filter is enabled, the output will be delayed up to four CLK cycles. Using these options, any output from the LUT shorter than two synchronized clock cycles will be filtered out.

Be aware that sometimes, based on the logic used as inputs to the LUT, a valid output signal can be high for a few clock cycles. If the filter option is chosen in such cases, it will break the function of the system by filtering out valid signals. The filter should only be used when it does not matter if the signal is delayed or shortened by the filter. Before implementing any of the filter options it would be wise to analyze what is the shortest valid signal out of the LUT in the current configuration. If the shortest signal is shorter than two cycles, a filter must not be used.

Figure 2-15. Filter

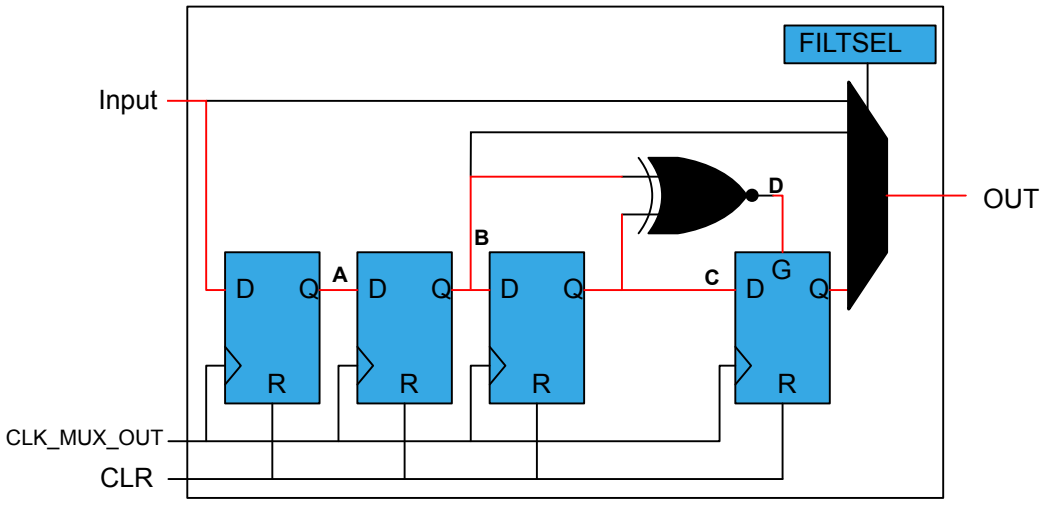
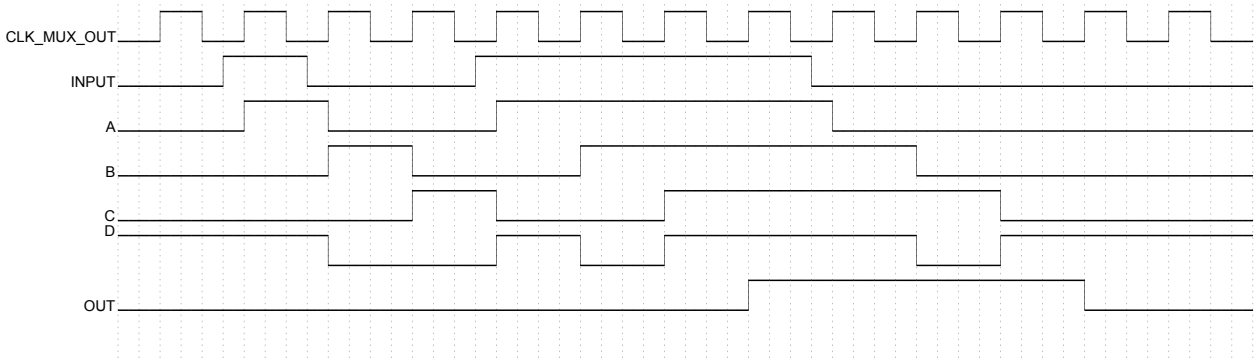


Figure 2-16. Filter timing



2.2.3 Edge Detector

The edge detector can be enabled to generate a pulse when a rising edge on the input is detected. To detect a falling edge, the truth table should be programmed to provide the opposite level. An example is to send a pulse with the event system to trigger another peripheral, e.g. a timer, every time the truth table has output HIGH(1).

Figure 2-17. Edge Detector

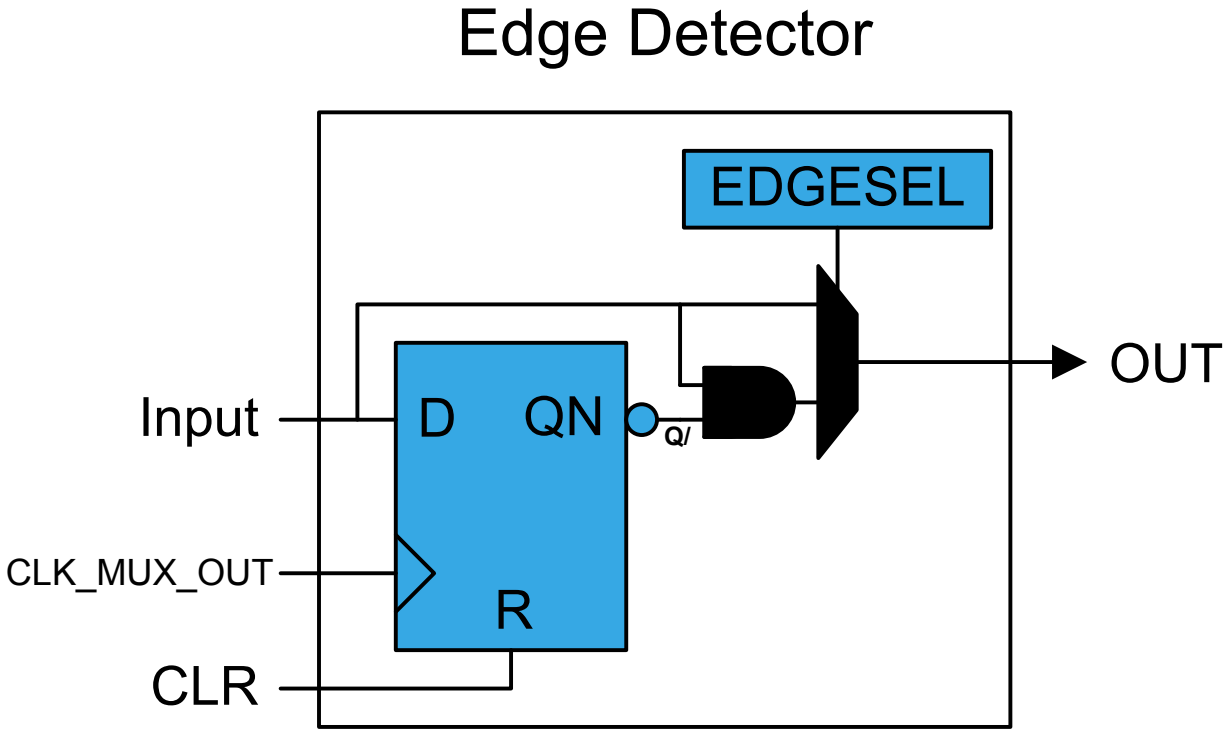
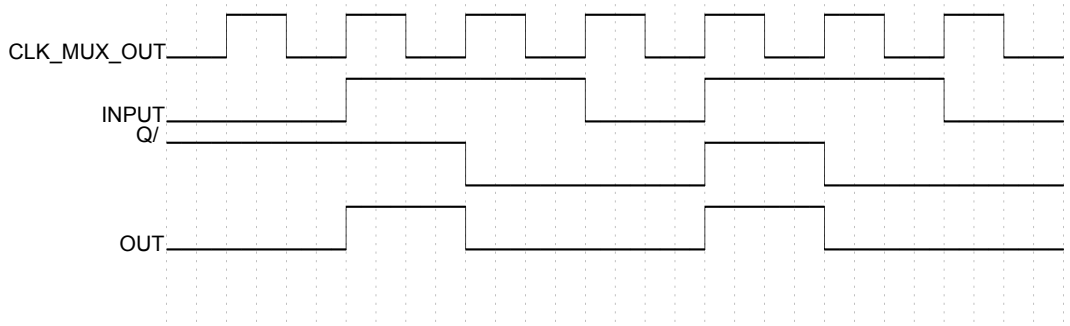


Figure 2-18. Edge Detector Timing



2.3 Sequential Logic

Each LUT pair can be connected to an internal sequential logic. The sequential selection bits, SEQSEL in the Sequential Control register, select between the different blocks available. Sequential logic can be used to achieve more complex functionality in the CCL.

The CCL has the following sequential logic blocks:

- Gated D Flip-Flop (DFF)
- JK Flip-Flop (JK)
- Gated D-Latch (DLATCH)
- RS Latch (RS)

In addition, a T Flip-Flop can be created using the JK Flip-Flop.

In Atmel Start it is possible to find application notes and code examples that use CCL and sequential logic. For example:

- AVR42779 Ultrasonic Distance Measurement

2.3.1 Gated D Flip-Flop

The D Flip-flop (DFF) is a widely used and is often called "data" or "delay" flip-flop. When G input is high, the flip-flop captures the value of the D-input and the captured value becomes the Q output. If the G input is low the D input is ignored and the Q output is unchanged from its last state The DFF can be seen as a memory cell, a zero-order hold, or a delay line.

The D-input is driven by the even LUT output (LUT0), and the G-input is driven by the odd LUT output (LUT1).

Figure 2-19. Gated D Flip-Flop

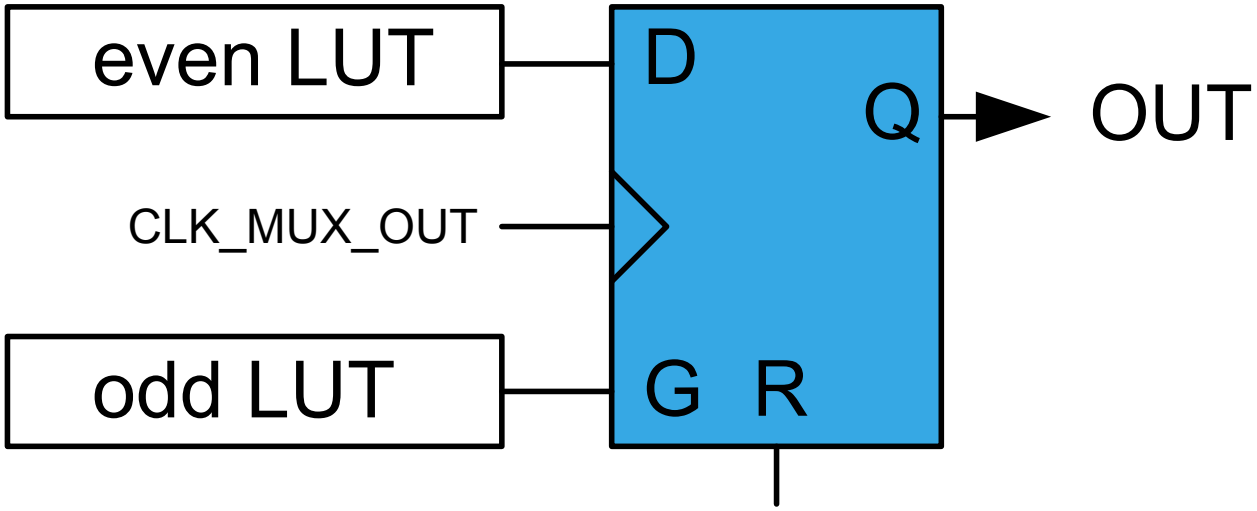


Table 2-12. DFF Behavior

R	G	D	OUT
1	X	X	Clear
0	1	1	Set
		0	Clear
	0	X	Hold state (no change)

2.3.2 JK Flip-Flop

The JK flip-flop is the most widely used of all the flip-flops and can be viewed as a universal flip-flop, since it can be configured to behave as an SR flip-flop, a D flip-flop, or a T flip-flop. It is basically a gated SR flip-flop without the illegal output states when J and K are equal or logic "1".

The J-input is driven by the even LUT output (LUT0), and the K-input is driven by the odd LUT output (LUT1).

Figure 2-20. JK Flip-Flop

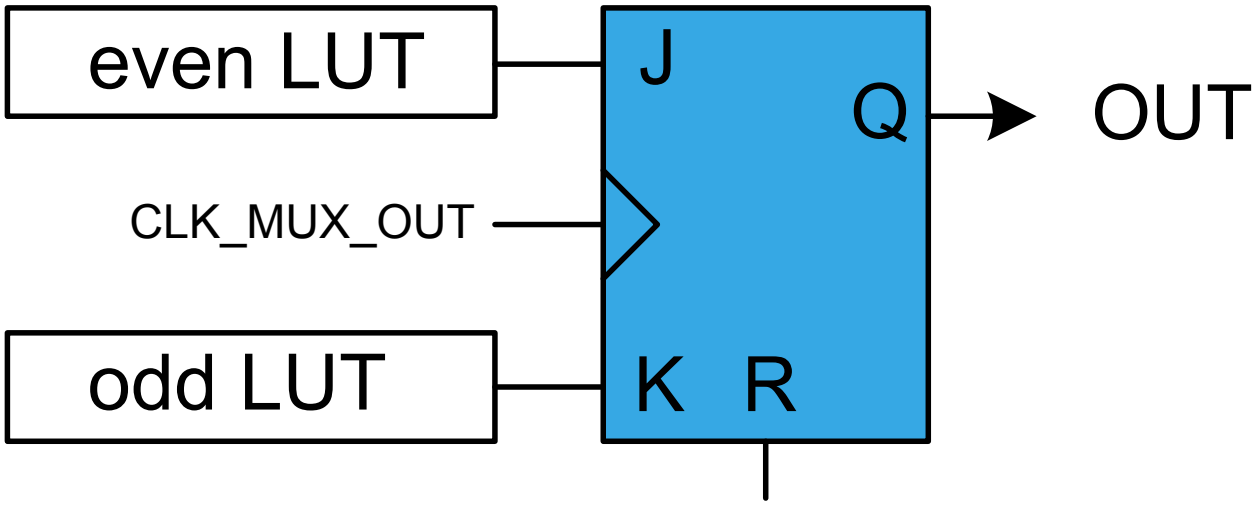


Table 2-13. JK Behavior

R	J	K	OUT
1	X	X	Clear
0	0	0	Hold state (no change)
0	0	1	Clear
0	1	0	Set
0	1	1	Toggle

2.3.3 Gated D-Latch

The D-latch is a multivibrator latch circuit without the “illegal” input state that the SR latch has when both inputs are high. The D-latch is known as a transparent latch. This means that as long as the gated signal G is high, the signal on D is propagated through the latch to the output.

The D-input is driven by the even LUT output (LUT0), and the G-input is driven by the odd LUT output (LUT1).

Figure 2-21. Gated D-Latch

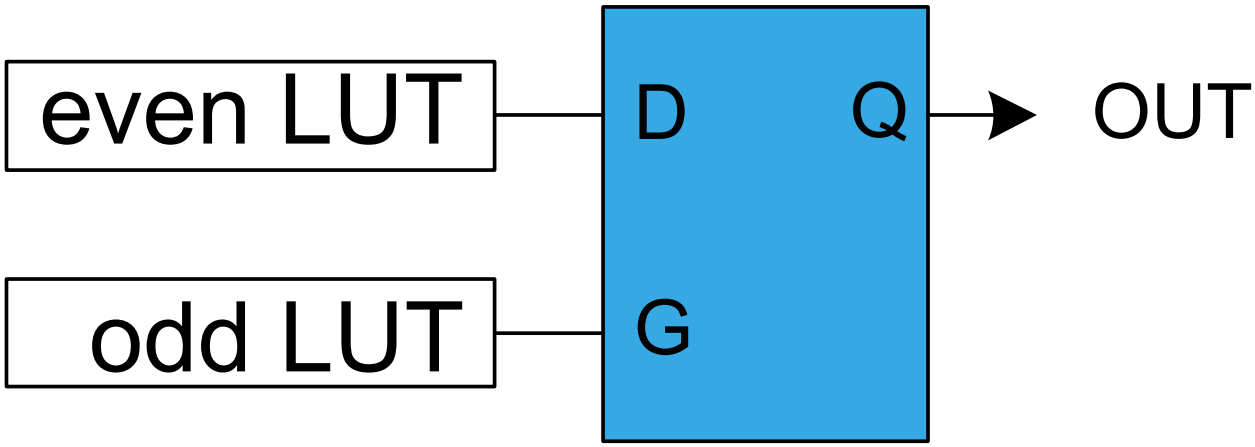


Table 2-14. D-Latch Behavior

G	D	OUT
0	X	Hold state (no change)
1	0	Clear
1	1	Set

2.3.4 RS Latch

The RS latch has basically the same functions as an SR latch, except in the forbidden state were both S and R equals 1. In this state an SR latch output will become "1", but on the RS the output will be "0".

The S-input is driven by the even LUT output (LUT0), and the R-input is driven by the odd LUT output (LUT1).

Figure 2-22. RS Latch

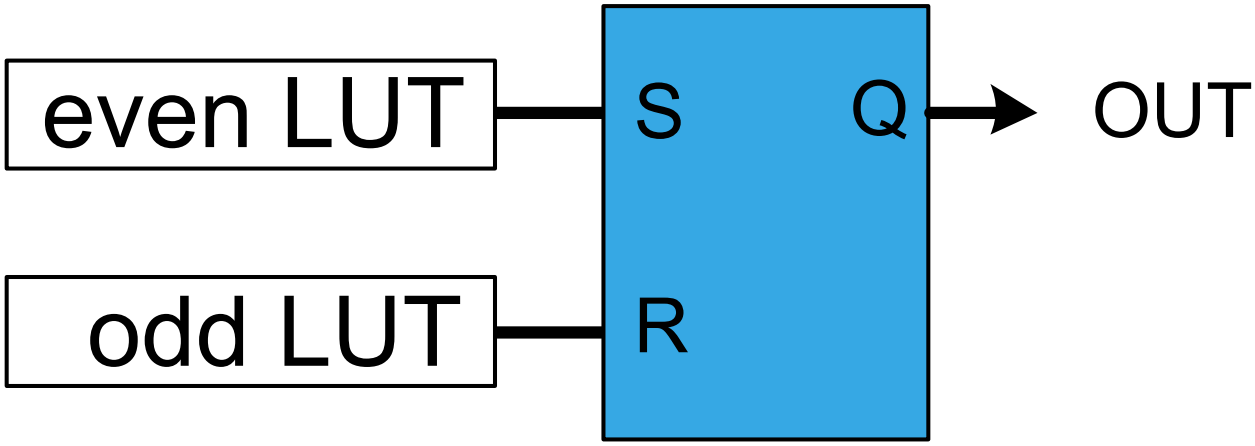


Table 2-15. RS Latch Behavior

S	R	OUT
0	0	Hold state (no change)
0	1	Clear
1	0	Set
1	1	Forbidden

2.3.5 T Flip-Flop

The T flip-flop, or toggle flip-flop, can be created by connecting both inputs on a JK flip-flop together it is possible to create a T flip-flop. This type of flip-flop can be used as a frequency divider. The T flip-flop will toggle the output on each clock cycle when both J and K inputs are high, so the output frequency will be half of the input frequency.

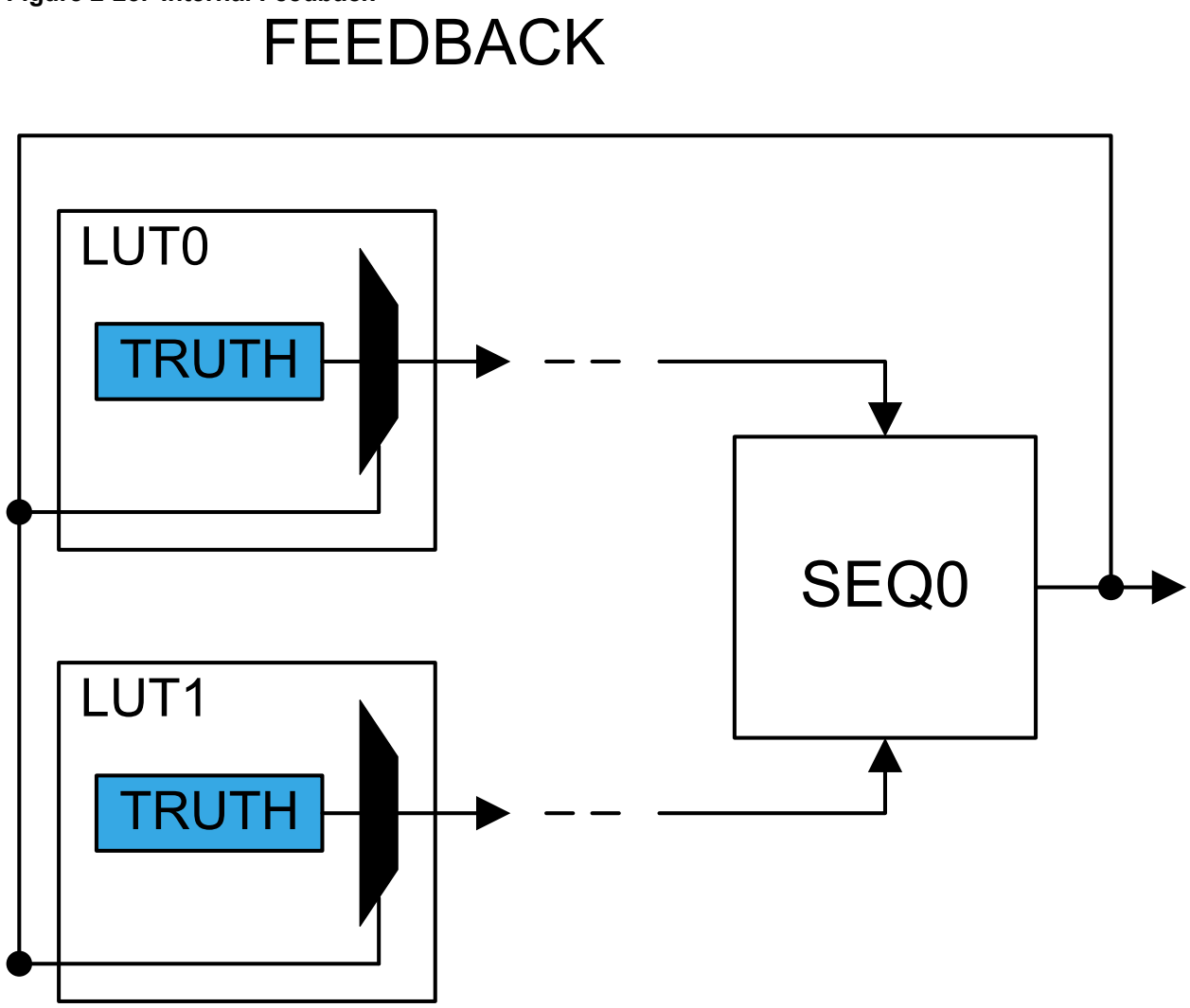
The filter and edge detector can be used to filter out any spikes that would cause the JK to toggle unintentionally.

2.3.6 Feedback

By feeding the output from the sequential logic back into the input of the LUT, a new type of device is created: the Finite State Machine (FSM).

In some systems it might be necessary to use feedback to achieve the desired functionality. Knowing the output state of the system can be very useful, so internal feedback from the sequential logic output is possible to any of the inputs on both LUTs, making the feedback system very flexible.

Figure 2-23. Internal Feedback



3. Introduction to Event System

The Event System (EVSYS) is a typical CIP, which allows a change in one peripheral (the Event generator) to trigger actions in other peripherals (the Event users) through Event channels. It is a simple but powerful system as it allows for autonomous control of peripherals without any use of interrupts, CPU, or DMA resources. It provides short and predictable response times between peripherals, and can reduce the complexity, size, and execution time of the software, and save power.

AVR usually supports several parallel Event channels, and one Event channel can be divided into three distinct parts:

- Event generators, with one or more Event sources
- The Event routing network
- Event users

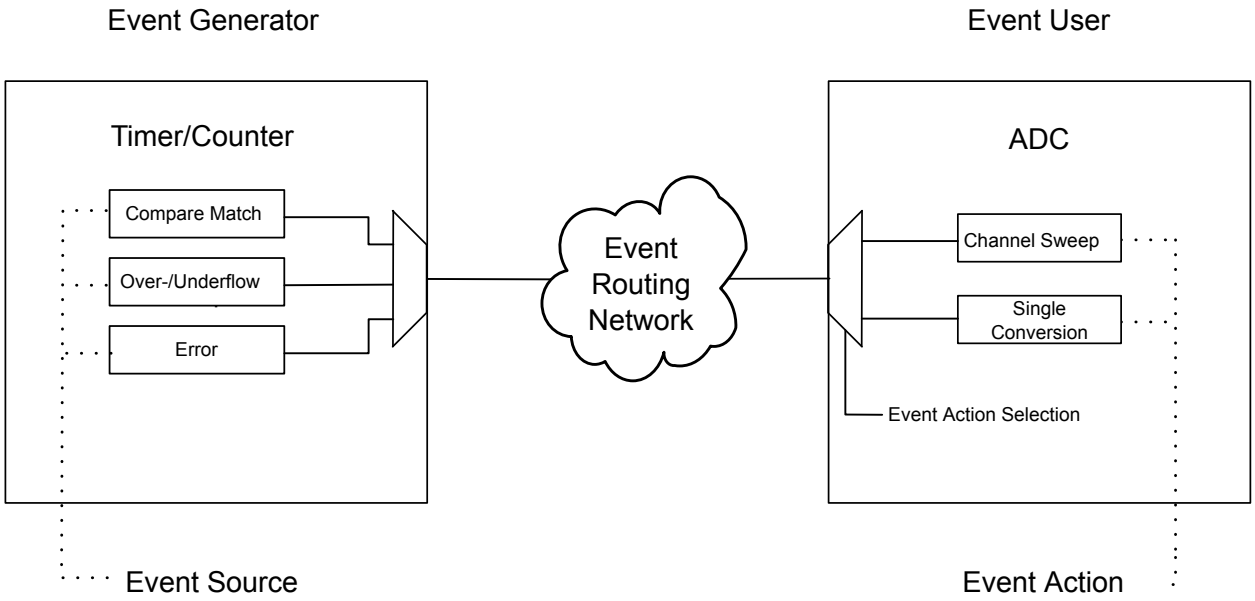
An Event is an indication that a change of state within a peripheral has occurred. A peripheral capable of generating Events is called an Event generator. One Event generator may be able to generate Events on several changes within the peripheral. Each of these is an individual Event source. A channel can be either asynchronous or synchronous to the main clock, based on the requirements of the application. For the tinyAVR® 1-Series, there are four asynchronous and two synchronous Event channels. Register ASYNCH0, ASYNCH1, ASYNCH2, ASYNCH3, SYCNCH0, and SYNCH1 are used to configure Event sources for these channels accordingly. Only one trigger from an Event generator peripheral can be routed on each Event channel, but multiple channels can use the same generator source. Multiple peripherals can use Events from the same channel.

The Event routing network handles the routing of Events from the Event generator to the Event user. Every Event source from every Event generator is connected to the inputs of each of the Event channels. An Event user is a peripheral module that can make use of an Event to trigger an action, referred to as an Event action. An Event user selects the Event source to react to by selecting an Event channel. The actual Event source is determined by the multiplier setting in the selected Event channel.

The Event system can directly connect analog and digital converters, analog comparators, I/O port pins, real-time counters, timer/counters, and the configurable custom logic peripherals. Events can also be generated from software and the peripheral clock.

The figure below shows a simplified version with one timer/counter as Event generator and one ADC as an Event user. The Event channel MUX's can select one of three available sources to be routed through the corresponding Event channel.

Figure 3-1. Example of Event Source, Generator, User, and Action



The Event system uses the peripheral clock for I/O registers and strobes. Also, it can be used in Sleep modes without any clock. An Event usually lasts for one clock cycle.

Manual Event Generation: It is possible to generate Events either from software or by using the on-chip debugging system. The generated Events are injected directly in the Event channels. The Event channel does not need to have an Event source associated with it to use the manual Event generation possibilities. If an Event source is associated with the Event channel, the manually generated Event has priority and will override the peripheral Event. Two registers are used for manual Event generation: STROBE and DATA. The Event generation is triggered by a write to the STROBE register. When generating signaling Events, only the STROBE register is needed. When generating data Events, both STROBE and DATA must be used and STROBE must be written after DATA.

Events and sleep modes: The Event system is operative in Active mode and Standby Sleep mode. In all other Sleep modes, peripheral modules will not be able to communicate using the Event system.

3.1 Overview of Event Features for Peripherals in the tinyAVR® 1-Series

Below is an overview of Event related features for peripherals in the tinyAVR® 1-Series, which are useful for developing core independent applications. Refer to the specific device data sheet for detailed information.

- PORT - I/O Pin Controller
  - Generate Events from all GPIO pins
- TCA - 16-bit Timer/Counter Type A
  - Count positive edges of Event signal
  - Count both edges of Event signal
  - Count prescaled clock cycles as long as the Event signal is high
  - Count prescaled clock cycles. Event signal controls the count direction.
  - Output Events can be generated based on counter overflow, underflow, and compare match
- TCB - 16-bit Timer/Counter Type B
  - Initialization, counting, and capture can be controlled by Event signal

  - For modes that generate output, the output can be distributed as an Event signal
- TCD - 12-bit Timer/Counter Type D
  - Output Events can be generated based on counter compare match
  - Output Events can be delayed by a configurable number of TCD delay clock cycles. The TCD delay clock is a prescaled version of the TCD clock.
  - Counter operation can be controlled in a number of different ways by two individual Event input signals
  - Possibility of masking and filtering input Events
- USART - Universal Synchronous and Asynchronous Receiver and Transmitter
  - Input Event signal can be used as receiver input instead of the corresponding RX pin
- RTC - Real Time Counter
  - Output Events can be generated on counter overflow and compare match
  - Output Events can be generated periodically corresponding to each n^{th} RTC clock period, where n is selectable from a predefined set of values
- CCL - Configurable Custom Logic
  - Each Lookup-table (LUT) can take two individual Events as inputs for its corresponding truth table
  - The output from each LUT can be distributed as Event signals
- AC - Analog Comparator
  - Comparator output can be distributed as an Event signal
- ADC - Analog to Digital Converter
  - Input Event can trigger an ADC conversion
- UPDI - Unified Program and Debug Interface
  - Generates an output Event that can be used to measure the system clock frequency

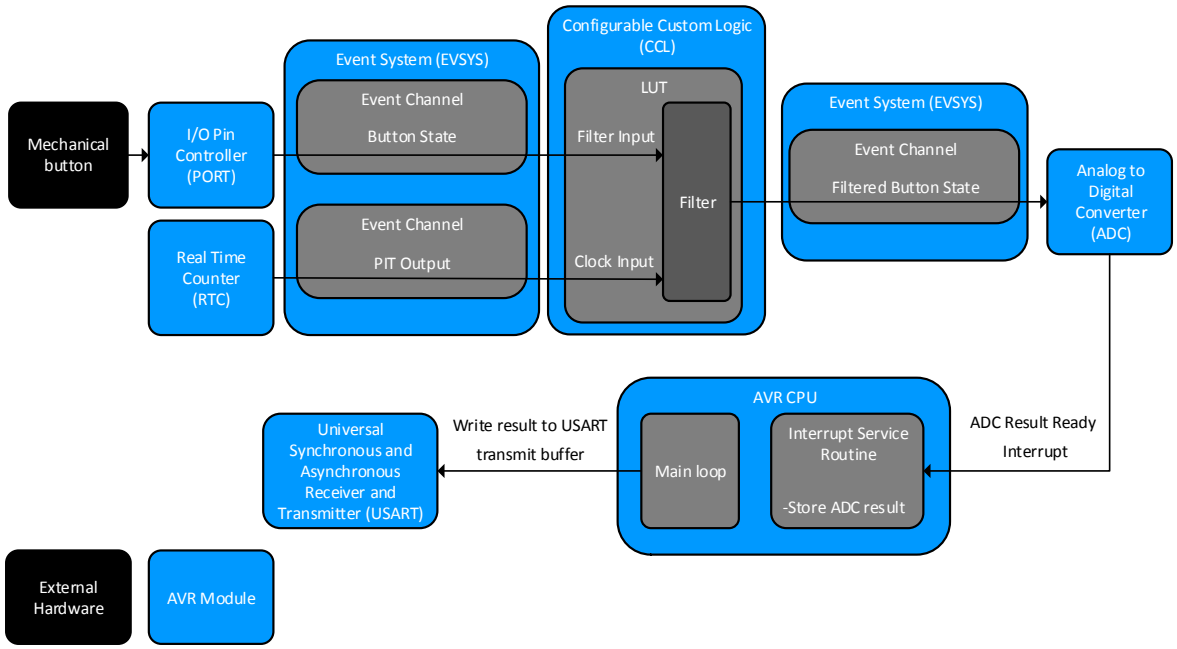
4. Application Example - Filtering Button Signal and Initiating ADC Conversion

Using the signal from a mechanical button directly into an application without any form of filtering will in many cases lead to unpredictable behavior since the signal often transitions several times between high and low each time the button is pushed or released. This is often referred to as *bounce*. If an application is required to act once each time a mechanical button is pressed, some form of filtering needs to be implemented either in hardware or software, also referred to as *debouncing*.

This chapter describes an application example that consistently initiates a single ADC conversion when a mechanical button is pressed, without involving the AVR core or adding external filtering. *Debouncing* the button signal is accomplished by filtering it with the CCL and using the filtered signal to trigger an ADC conversion. The signals are routed via the Event System, and when the conversion result is ready the result is transmitted via the USART module for verification.

The figure below shows an overview of how the utilized device modules, the CPU, and the connections between them are configured. For details on how the application is implemented on a specific device or evaluation kit, open and inspect the example application in Atmel START. How to find the application in Atmel START is described in chapter Get Source Code from Atmel | START.

Figure 4-1. Example Overview



4.1 Event System (EVSYS) Setup

The application example uses the Event System to route the signals to and from the CCL for maximum flexibility. The button signal and a suitable clock signal must be routed to the Event inputs of a LUT, while the output from this LUT must be routed to the ADC Event input. Therefore, in this application the CCL will be both an Event generator and an Event user.

The Event output from the Periodic Interrupt Timer (PIT) unit in the Real-Time Counter (RTC) is suitable as a clock signal, and by using it, other timer/counters on the device are kept available for other purposes. If the RTC clock is set to 32kHz, a good starting point would be to select the PIT output Event corresponding to dividing the RTC clock by 1024 as the source for one Event channel. This might need to be modified depending on the characteristics of the button signal. The input Event selected as IN[2] for the LUT should then be configured to be a user of this channel.

The I/O pin connected to the button should be configured as the Event generator for a second Event channel. The remaining available LUT input Event should then be configured as a user of this channel. The I/O pin should also be configured as an input with its associated pull-up resistor enabled if there is no external pull-up resistor connected.

To trigger an ADC conversion from the filtered button signal, the LUT output should be configured as the generator for a third Event channel, while the ADC should be configured as a user.

4.2 Real Time Counter (RTC) Setup

The RTC module includes a function called PIT. The PIT uses the same clock source as the rest of the RTC and when enabled, provides a set of output events in the form of clock signals with periods corresponding to *n* times the RTC clock period. The different PIT output events are selectable in the Event System in the form of a set of predefined event generators, each with a different period relative to the RTC clock.

In order to use the PIT output events the PIT must be enabled in the RTC module.

4.3 Configurable Custom Logic (CCL) Setup

Each Look-Up Table (LUT) in the CCL includes a filter that can be used to synchronize or filter the LUT output. The filter is by default clocked by the peripheral clock signal, but an alternative clock signal provided to the LUT on IN[2] can be used. By providing a suitable clock signal on IN[2] and the signal from a mechanical button on either IN[0] or IN[1], a single LUT can be used to filter glitches on the button signal that would otherwise cause unwanted behavior.

To configure a LUT for this purpose, its filter and alternative clock source features must be enabled.

The LUT inputs can be selected from a large number of different signals, among them two different Event signals. For maximum flexibility in terms of sources for the button and clock signal, the two Event signals should be selected as inputs. One of the Event inputs must be assigned to IN[2] to be used as an alternative clock signal. The other Event signal should be assigned to one of the two remaining inputs, while the unused input should be configured as *Masked*.

Since IN[2] will be masked as well when the alternative clock feature is enabled, only the input selected for the button signal needs to be considered when configuring the TRUTH register of the LUT. The LUT output should be high as long as the button is pressed. For instance, if the button signal is active high and available on IN[1], the TRUTH register should be set to 4. If the button signal is active low, which is the case for many evaluation kits, the TRUTH register should be set to 1.

Complete the CCL setup by enabling the LUT and the CCL.

Signals from I/O pins and/or other peripherals can be selected as LUT inputs instead of Event signals, if required by the application.

4.4 Analog-to-Digital Converter (ADC) Setup

For the application to be able to initiate ADC conversions from an Event signal instead of using the core, the Start Event Input - feature of the ADC must be enabled. Then, to store and handle the conversion result as soon as it is available, the result ready interrupt should be enabled as well.

One of the internal analog sources available to the ADC is the voltage from the on-board temperature sensor. To configure the ADC to sample the temperature sensor, the ADC reference should be set to the internal reference, and the sensor should be selected as the ADC input signal. Then, the ADC voltage reference should be set to 1.1V and enabled in the Voltage Reference (VREF) module.

By setting the ADC up as described, a 10-bit converted voltage value will be available in the ADC Result register when the result ready interrupt is requested. To convert the result to a temperature value, it must be corrected by an offset and a gain factor included in the signature row of the device. For simplicity, this correction is not included in the application example.

4.5 Universal Synchronous and Asynchronous Receiver and Transmitter (USART) Setup

For verification and testing purposes it can be helpful to transmit data to a serial terminal for visualization. To configure the USART to send data over its TX (transmit) pin, it is required to only enable the transmitter, set the baud rate, and configure the USART TX pin as an output. By using the USART driver provided by Atmel START the baud rate is calculated and configured by the driver.

4.6 CPU Details

Since the result ready interrupt is enabled in the ADC and the application example should store and transmit ADC results via the USART, the correct Interrupt Service Routine (ISR) should be implemented along with a mechanism to forward data to the USART.

The result ready interrupt routine could be implemented similarly to the snippet below, given that the variables `ADC_result` and `send_flag` have been defined.

```
ISR(ADC0_RESRDY_vect)
{
    /* Store the ADC result and notify the main loop to send the result */
    ADC_result = ADC0.RESL;
    send_flag  = 1;

    /* The interrupt flag has to be cleared manually */
    ADC0.INTFLAGS = ADC_RESRDY_bm;
}
```

For simplicity, the example only stores and transmits the eight least significant bits of the ADC result.

Transmission of the stored value using a USART driver function generated by Atmel START can then be implemented in the main loop in a similar way as in the snippet below.

```
/* ADC result has been stored and is ready to be sent */
if (send_flag) {
    USART_0_putc(ADC_result);
    send_flag = 0;
}
```

The `USART_0_putc()` function simply writes the given eight bits to the USART transmit register.

To enable interrupts globally on the device the I-bit in the CPU Status Register (SREG) must be set as well.

5. Get Source Code from Atmel | START

The example code is available through Atmel | START, which is a web-based tool that enables configuration of application code through a Graphical User Interface (GUI). The code can be downloaded for both Atmel Studio 7.0 and IAR Embedded Workbench® via the direct example code-link(s) below or the *BROWSE EXAMPLES* button on the Atmel | START front page.

Atmel | START web page: http://microchip.com/start

Example Code

- Getting Started with Core Independent Peripherals:
  - http://start.atmel.com/#example/Atmel%3Agetting_started_with_core_independent_peripherals%3A1.0.0%3A%3AApplication%3AGetting_Started_with_Core_Independent_Peripherals%3A

Press *User guide* in Atmel | START for details and information about example projects. The *User guide* button can be found in the example browser, and by clicking the project name in the dashboard view within the Atmel | START project configurator.

Atmel Studio

Download the code as an .atzip file for Atmel Studio from the example browser in Atmel | START, by clicking *DOWNLOAD SELECTED EXAMPLE*. To download the file from within Atmel | START, click *EXPORT PROJECT* followed by *DOWNLOAD PACK*.

Double-click the downloaded .atzip file and the project will be imported to Atmel Studio 7.0.

IAR Embedded Workbench

For information on how to import the project in IAR Embedded Workbench, open the Atmel | START user guide, select *Using Atmel Start Output in External Tools*, and *IAR Embedded Workbench*. A link to the Atmel | START user guide can be found by clicking *About* from the Atmel | START front page or *Help And Support* within the project configurator, both located in the upper right corner of the page.

6. Other Relevant Resources

Below is an overview of application notes and Atmel START example projects utilizing core independent peripherals.

Table 6-1. Atmel START Example Projects

Application note	Link
Core Independent Nightlight Using Configurable Custom Logic on ATtiny1617	http://www.microchip.com/wwwappnotes/appnotes.aspx?appnote=en595063
Core Independent Brushless DC Fan Control Using Configurable Custom Logic on ATtiny817	http://www.microchip.com//wwwAppNotes/AppNotes.aspx?appnote=en592093
Digital Sound Recorder using DAC with ATtiny817	http://www.microchip.com//wwwAppNotes/AppNotes.aspx?appnote=en592092
Core Independent Ultrasonic Distance Measurement with ATtiny817	http://www.microchip.com//wwwAppNotes/AppNotes.aspx?appnote=en592094

Table 6-2. Atmel START Example Projects

Example	Link
Getting STARTed AVR Events	http://start.atmel.com/#example/Atmel%3AApplication_AVR_Examples%3A1.0.0%3A%3AApplication%3AGetting_STARTed_AVR_Events%3A
Digital Sound Recorder	http://start.atmel.com/#example/Atmel%3Avoice_recorder_with_dac%3A1.0.0%3A%3AApplication%3AAVR42777_Digital_Sound_Recorder%3A
Parrot	http://start.atmel.com/#example/Atmel%3Aparrot_feg%3A1.0.0%3A%3AApplication%3AAVR42777_Parrot%3A
BLDC Fan Control	http://start.atmel.com/#example/Atmel%3Aavr42778_bldc_fan_control%3A1.0.0%3A%3AApplication%3AAVR42778_BLDC_Fan_Control%3A
Ultrasonic Distance Measurement	http://start.atmel.com/#example/Atmel%3Acip_ultrasonic_distance%3A1.0.0%3A%3AApplication%3AAVR42779_Ultrasonic_Distance_Measurement%3A
Using ATtiny817 Event System	http://start.atmel.com/#example/Atmel%3Aavr42815_using_event_system_on_attiny817%3A0.0.1%3A%3AApplication%3AAVR42815_-_Using_ATtiny817_Event_System%3A

Example	Link
Core Independent Night Light Using CCL	http://start.atmel.com/#example/Atmel%3Acore_independent_night_light_using_ccl%3A1.0.0%3A%3AApplication%3ACore_Independent_Night_Light_using_CCL%3A
Quadrature decoding using CCL with TCA and TCB	http://start.atmel.com/#example/Atmel%3Aquadrature_decoding_using_ccl_with_tca_and_tcb%3A1.0.0%3A%3AApplication%3AQuadrature_Decoding_using_CCL_with_TCA_and_TCB%3A
Realistic Heartbeat	http://start.atmel.com/#example/Atmel%3Acip_realistic_heartbeat%3A1.0.0%3A%3AApplication%3ARealistic_Heartbeat%3A

7. Revision History

Doc Rev.	Date	Comments
A	04/2017	Initial document revision.
B	02/2018	Chapter *Relevant Devices* has been updated to include tinyAVR 0-series and megaAVR 0-series

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.

- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-2718-6

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
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	**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	**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	**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: 47-7289-7561 **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