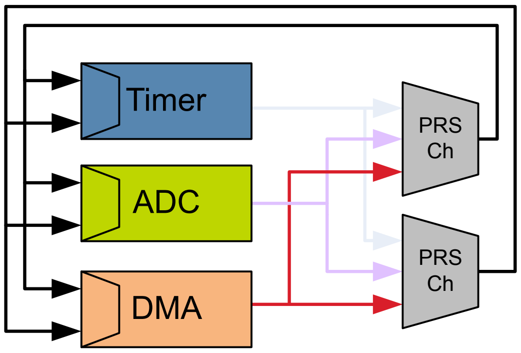
AN0025: Peripheral Reflex System (PRS)

This application note describes the Peripheral Reflex System (PRS) features and how these can be used to improve the system's energy performance and reduce MCU workload and latency.

KEY POINTS

- This document discusses PRS producers and consumers.
- Asynchronous mode PRS for EM2/3 operation.
- The enhanced PRS supports configurable logic.
- This application note includes:
  - This PDF document
  - Source files
    - Example C-code
    - Multiple IDE projects



1. Device Compatibility

This application note supports multiple device families, and some functionality is different depending on the device.

MCU Series 0 consists of:

- EFM32G
- EFM32GG
- EFM32WG
- EFM32LG
- EFM32TG
- EFM32ZG
- EFM32HG

Wireless MCU Series 0 consists of:

- EZR32WG
- EZR32LG
- EZR32HG

MCU Series 1 consists of:

- EFM32JG1/EFM32JG12
- EFM32PG1/EFM32PG12
- EFM32GG11/EFM32GG12
- EFM32TG11

Wireless SoC Series 1 consists of:

- EFR32BG1/EFR32BG12/EFR32BG13
- EFR32FG1/EFR32FG12/EFR32FG13/EFR32FG14
- EFR32MG1/EFR32MG12/EFR32MG13/EFR32MG14

MCU Series 2 consists of:

- EFM32PG22
- EFM32PG23
- EFM32PG26
- EFM32PG28

Wireless SoC Series 2 consists of:

- EFR32BG21/EFR32MG21
- EFR32BG22/EFR32BG22L/EFR32FG22/EFR32MG22
- EFR32FG23/EFR32SG23/EFR32ZG23
- EFR32BG24/EFR32BG24L/EFR32MG24
- EFR32FG25
- EFR32BG26/EFR32MG26
- EFR32BG27/EFR32MG27
- EFR32FG28/EFR32SG28/EFR32ZG28

2. Peripheral Reflex System

2.1 Introduction

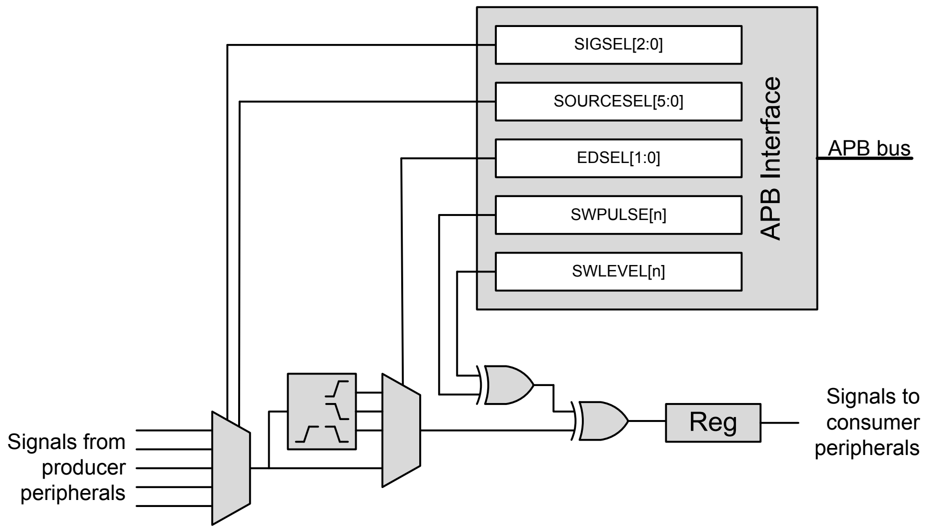
The PRS is a signal routing network allowing direct communication between different peripheral modules without involving the CPU. Peripheral modules which send out reflex signals to the PRS are called producers, and modules accepting reflex signals are called consumers. The PRS routes the reflex signals from producer to consumer peripherals, which perform actions depending on the reflex signals received.

2.2 Overview

The PRS module contains certain interconnect channels (see the device reference manual for the available PRS channels), and each of these can select between all the output reflex signals offered by the producers. The consumers can then choose which PRS channel to listen to and perform actions based on the reflex signals routed through that channel. The reflex signals can be both pulse signals and level signals.

An overview of the PRS reflex channel for Series 0 and Series 1 devices is shown in Figure 2.1 PRS Channel Overview for Series 0 and Series 1 devices on page 4 and an overview of the asynchronous PRS reflex channel for Series 2 devices is shown in Figure 2.2 PRS Asynchronous Channel Overview for Series 2 devices on page 5. On Series 2 devices, synchronous channels are similar to asynchronous channels but do not include the configurable logic block or SWLEVEL / SWPULSE features.

Figure 2.1. PRS Channel Overview for Series 0 and Series 1 devices



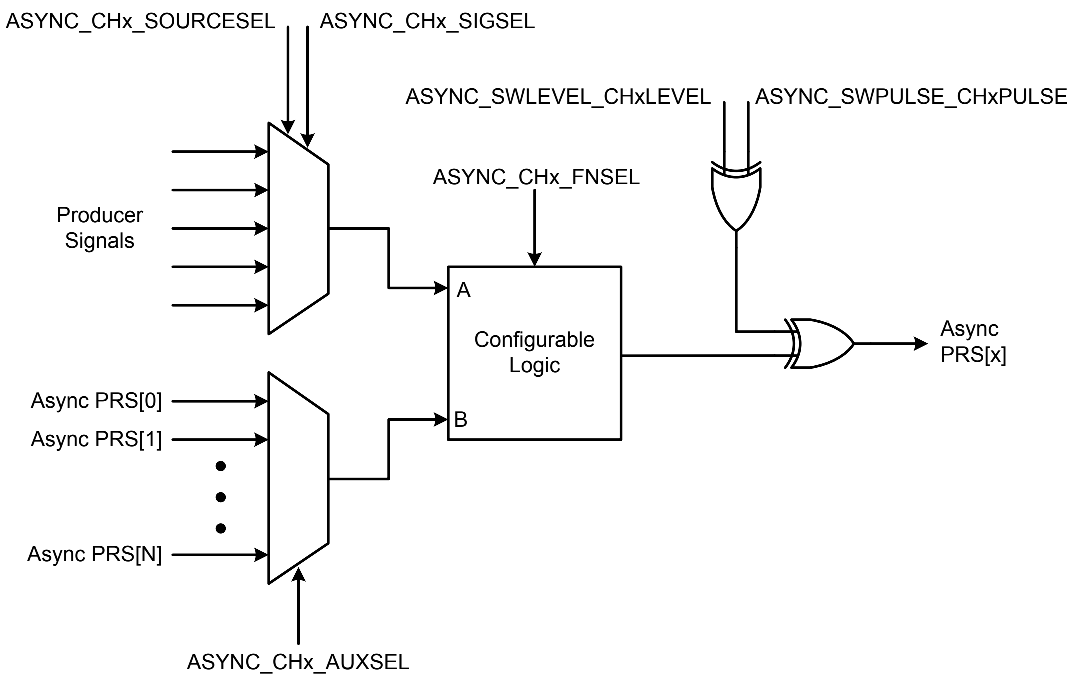


Figure 2.2. PRS Asynchronous Channel Overview for Series 2 devices

3. General Operation

3.1 Asynchronous Mode

Reflex channels can operate in two modes: synchronous or asynchronous. In synchronous mode, reflex signals are clocked on high-frequency clocks. As a result, synchronous channels are restricted to EM0 and EM1 power modes. In contrast, asynchronous channels remain operational across EM0, EM1, EM2, and EM3 power modes.

For Series 0 and Series 1 devices:

- Asynchronous reflexes have certain limitations, as they can only be utilized by a subset of the available reflex consumers marked with Async Support in Table 3.3 Reflex Consumers for Series 0 and Series 1 Devices on page 11. Peripherals capable of generating asynchronous reflex signals are also specifically marked with async support in the Table 3.2 Reflex Producers for Series 0 and Series 1 Devices on page 9.
- To use these reflex signals asynchronously, the ASYNC bitfield in the PRS_CHx_CTRL register must be set for the corresponding PRS channel selecting the reflex signal.
- Among the consumers that can take advantage of asynchronous reflex signals are the LESENSE, LEUART, and PCNT modules. Additionally, the USART/UART can utilize certain asynchronous signals, though the specifics vary. For further details on supported asynchronous signals, refer to the respective device reference manual.

Note:

1. Asynchronous mode is not available in the EFM32 Gecko (EFM32G) device.
2. On Series 0 and Series 1 devices, if a reflex channel with ASYNC set is used in a consumer that does not support asynchronous reflexes, the behavior is undefined.

For Series 2 devices:

- Asynchronous channels offer greater flexibility, as they can be connected to any signal provided by producers.
- In contrast, synchronous channels are restricted to specific signals originating from the TIMER, IADC, and VDAC modules. On the consumer side, all peripherals can listen to asynchronous channels, while only TIMER, IADC, and VDAC modules can listen to synchronous channels.

3.2 Channel Functions

Different functions can be applied to a reflex signal within the PRS module. It is also possible to generate output reflex signals by configuring the SWPULSE and SWLEVEL bits.

On Series 0 and Series 1 devices, each channel includes an edge detector to enable generation of pulse signals from level signals. Edge detection can be applied to a PRS signal using the EDSEL bitfield in the PRS_CHx_CTRL register. When edge detection is enabled, changes in the PRS input will result in a pulse on the PRS channel.

For Series 1 devices:

- Signals on the PRS input also have to be at least one HFBUSCLK period wide to be detected properly. This applies to all cases when asynchronous mode is not used in the PRS.
- There are two options for communication between peripherals on different prescaled clocks, for example between peripherals on HFBUSCLK and HFPERCLK:
  - For level signals, no action is needed, but software must make sure that the level signals are held long enough for the destination domain to detect them.
  - For pulse signals, edge detection and stretch (EDSEL and STRETCH bitfields in the PRS_CHx_CTRL register) should be enabled. When edge detection and stretch are enabled on a PRS source, the output on the PRS channel is held long enough for the destination domain to detect the pulse. This also works if there are multiple destination domains running at different frequencies.

On Series 0 and Series 1 devices, PRS channels can be manually triggered by writing to the PRS_SWPULSE or PRS_SWLEVEL registers. On Series 2 devices, only asynchronous PRS channels can be manually triggered using the PRS_ASYNC_SWPULSE or PRS_ASYNC_SWLEVEL registers, as synchronous mode on these devices does not include the configurable logic block or SWLEVEL/SWPULSE features.

SWLEVEL is a programmable level for each channel and holds the value it is programmed to. The SWPULSE will cause the PRS channel to output a high pulse lasting one clock cycle, determined by the high frequency system clock: HFPERCLK for Series 0 and 1 devices, HFBUSCLK for Series 1 devices, or EM01GRPACLK for Series 2 devices. This pulse is produced when the corresponding channel bit is set to 1, otherwise a 0 is asserted.

The SWLEVEL and SWPULSE signals are then XOR'ed with the output from the configurable logic block to form the output signal, which is sent to the channel selection logic for every consumer signal. For example, when SWLEVEL is set, if configurable logic produces a signal of 1, this will cause a channel output of 0. This is illustrated in Figure 2.1 PRS Channel Overview for Series 0 and Series 1 devices on page 4 and Figure 2.2 PRS Asynchronous Channel Overview for Series 2 devices on page 5.

The emlib functions:

- `void PRS_SourceSignalSet(unsigned int ch, uint32_t source, uint32_t signal, PRS_Edge_TypeDef edge)`
- `void PRS_SourceAsyncSignalSet(unsigned int ch, uint32_t source, uint32_t signal)`

can be used to easily configure the PRS channels.

By specifying the PRS channel, producing peripheral, signal from the peripheral, and edge for pulse generation (applicable to synchronous mode on Series 0 and 1 devices), the function configures the PRS accordingly.

Note: On Series 0 and Series 1 devices, the edge detector controlled by the EDSEL bitfield in the PRS_CHx_CTRL register should only be used when working with synchronous reflexes, that is, the ASYNC bitfield in the PRS_CHx_CTRL register is cleared.

3.3 Route PRS Channel to GPIO Pin

It is possible to route a PRS channel to a GPIO pin for debugging or use it as an enable signal.

Table 3.1. Route PRS Channel to GPIO pin

	EFM32 Gecko Series 0 and EZR32 Series 0	EFM32 Gecko Series 1 and EFR32 Wireless Gecko Series 1	EFM32 Gecko Series 2 and EFR32 Wireless Gecko Series 2
Availability	Only PRS channel 0 to 3 can route to GPIO pin	All PRS channels can route to GPIO pin	All PRS channels can route to GPIO pin
Register	PRS_ROUTE register for enable and location	• PRS_ROUTE register for enable • PRS_ROUTELOCx registers for location	• GPIO_PRSn_ROUTEEN register for enable • GPIO_PRSn_ASYNCHxROUTE or GPIO_PRSn_SYNCHxROUTE registers for pin/port selection

Note: This feature is not available in the EFM32 Gecko (EFM32G) device.

3.4 Producers

Each PRS channel can choose between signals from several producers, which are configured in the SOURCESEL bitfield in the PRS_CHx_CTRL register (for Series 0/1 devices) or PRS_SYNCHx_CTRL/PRS_ASYNCHx_CTRL register (for Series 2 devices). Each producer outputs one or more signals which can be selected by setting the SIGSEL field. Setting the SOURCESEL bits to 0 (Off) leads to a constant 0 output from the input mux regardless of SIGSEL. An overview of the available producers is given in the table below.

Listed below are the Reflex Producers for Series 0 and Series 1 devices. For a complete list of Synchronous and Asynchronous PRS Reflex Producers for Series 2 devices, consult the device reference manual.

Note: Availability of modules and reflex outputs may vary depending on the specific device. For detailed information, refer to the device reference manual and data sheet.

Table 3.2. Reflex Producers for Series 0 and Series 1 Devices

Module	Reflex Output	Output Format	Async Support
ACMP	Comparator Output	Level	Yes
ADC	Single Conversion Done	Pulse	Yes[1]
	Scan Conversion Done	Pulse	Yes[1]
DAC	Channel 0 Conversion Done	Pulse	—
	Channel 1 Conversion Done	Pulse	—
GPIO	Pin 0 to Pin 15 Input	Level	Yes
RTC	Overflow	Pulse	Yes
	Compare Match 0	Pulse	Yes
	Compare Match 1	Pulse	Yes
TIMER	Underflow	Pulse	—
	Overflow	Pulse	—
	Compare/Capture Channel Output	Level	—
UART	TX Complete	Pulse	—
	RX Data Received	Pulse	—
USART	TX Complete	Pulse	—
	RX Data Received	Pulse	—
	IrDA Decoder Output	Level	—
	RTS Output	Level	—
	TX Output	Level	—
	CS Output	Level	—
VCMP	Comparator Output	Level	Yes
LETIMER	CH0 Output	Level	Yes
	CH1 Output	Level	Yes
LESENSE	SCANRES Register	Level	Yes
	Decoder Output	Level/Pulse	Yes

Module	Reflex Output	Output Format	Async Support
BURTC	Overflow	Pulse	Yes
	Compare Match 0	Pulse	Yes
USB	Start of Frame	Level	Yes[2]
	Start of Frame Sent/Received	Level	Yes[2]
PRS	CH0 to CHx (Maximum Channel) Output	Level/Pulse	Yes
RTCC	CCV0 Output	Level/Pulse	Yes
	CCV1 Output	Level/Pulse	Yes
	CCV2 Output	Level/Pulse	Yes
PCNT	Triggered Compare Match	Level	Yes
	Overflow	Pulse	Yes
	Underflow	Pulse	Yes
	Direction	Level	Yes
CRYOTIMER	PERIOD Output	Pulse	Yes
CMU	Clock Output 0	Level	Yes
	Clock Output 1	Level	Yes

Note:

1. The Async Support for ADC is not available in EFM32 Gecko Series 0 and EZR32 Series 0 devices.
2. USB PRS outputs must be synchronized in the PRS when used, that is, it is an asynchronous PRS output. The USB PRS outputs go to 0 in Energy Mode 2 (EM2) or Energy Mode 3 (EM3).

3.5 Consumers

Consumer peripherals can be set to listen to a PRS channel and perform an action based on the signal received on that channel. Most consumers expect a pulse input, while some can handle level inputs as well.

Listed below are the Reflex Producers for Series 0 and Series 1 devices.

Note: Availability of modules and reflex outputs may vary depending on the specific device. For detailed information, refer to the device reference manual and data sheet.

Table 3.3. Reflex Consumers for Series 0 and Series 1 Devices

Module	Reflex Input	Input Format	Async Support
ADC	Single Mode Trigger	Pulse	Yes[1]
	Scan Mode Trigger	Pulse	Yes[1]
DAC	Channel 0 Trigger	Pulse	—
	Channel 1 Trigger	Pulse	—
TIMER	Compare/Capture Channel Input	Pulse/Level	—
	Alternate Input for DTI	Level	—
	Alternate Input for DTI Fault 0	Level	—
	Alternate Input for DTI Fault 1	Level	—
UART	TX/RX Enable	Pulse	—
	Alternate Input for RX	Level	Yes
USART	TX/RX Enable	Pulse	—
	Alternate Input for IrDA	Level	—
	Alternate Input for RX	Level	Yes
	Alternate Input for CLK	Level	Yes
LESENSE	Start Scan	Pulse/Level	Yes
	Decoder Bit 0 to 3	Level	Yes
IDAC	Alternate Input for OUTMODE	Level	Yes
LEUART	Alternate Input for RX	Level	Yes
PCNT	Compare/Clear Trigger	Pulse/Level	Yes
	Alternate Input for S0IN	Level	Yes
	Alternate Input for S1IN	Level	Yes
WDOG	Peripheral Watchdog	Pulse	Yes
LETIMER	Start LETIMER	Pulse	Yes
	Stop LETIMER	Pulse	Yes
	Clear LETIMER	Pulse	Yes
RTCC	Compare/Capture Channel Input	Pulse/Level	Yes

Module	Reflex Input	Input Format	Async Support
PRS	Set Event	Pulse	—
	DMA Request 0	Pulse	—
	DMA Request 1	Pulse	—
CMU	Alternate Input for Calibration Up-Counter	Level	—
	Alternate Input for Calibration Down-Counter	Level	—

Note:

1. The Async Support for ADC is not available in EFM32 Gecko Series 0 and EZR32 Series 0 devices.

For Series 2 devices:

- Consumer peripherals can be configured to listen to a PRS channel and perform an action based on the signal received on that channel. This is done by programming the PRSSEL or SPRSSEL in the consumer registers. SPRSSEL is only present for signals with the ability to listen to synchronous channels.
- The consumer registers follow the naming convention PRS_CONSUMER_<peripheral_name>_<signal_name>. For example, the PRS_CONSUMER_TIMER0_CC0 register is used to select which PRS channel output is sent to the TIMER0 peripheral's CC0 signal. In turn, the target peripheral should be configured to use the associated PRS trigger as desired. This is described in the individual peripheral chapters.
- PRS signals can be routed to GPIO pins by setting the GPIO_PRSn_ROUTEEN and GPIO_PRSn_ASYNCHxROUTE or GPIO_PRSn_SYNCHxROUTE registers.

Note: When configuring the synchronous PRS consumer registers, the target peripheral should be disabled or configured to not use the affected PRS signal. This will ensure that no false triggers occur at the consumer.

4. Advanced Features

These advanced features are not available in EFM32 Gecko Series 0 and EZR32 Series 0 devices.

4.1 Configurable PRS Logic

The configurable logic feature enables a PRS channel to perform logic operations on the signal coming from the selected producer.

Series 1

Each PRS channel has three logic functions that can be used by themselves or in combination. The selected PRS source can be AND'ed with the next PRS channel output, OR'ed with the previous PRS channel output, and inverted. The order of the functions is important. If OR and AND are enabled at the same time, AND is applied first, and then OR. Note that the previous and next channel options wrap around. Using the ORPREV option on the first PRS channel ORs with the output of the last PRS channel. Likewise, using the ANDNEXT option on the last PRS channel ANDs with the output of the first PRS channel

In addition to the logic functions that can combine a PRS channel with one of its neighbors, a PRS channel can also select any other PRS channel as the input. This can allow relatively complex logic functions to be created.

Series 2

Every asynchronous channel has a configurable logic block that can be programmed using the FNSEL field in the asynchronous channel control register. The configurable logic block for each channel has two inputs. Input A is the signal from the selected producer determined by SOURCESEL and SIGSEL of PRS_ASYNC_CHn_CTRL. Input B may be selected from the output of any other asynchronous PRS channel using the ASYNC_CHx.AUXSEL field. This allows for more complex logic functions to be created using multiple PRS channels.

The configurable logic feature is implemented as a 2 input look up table, with each bit of FNSEL representing the outcome for a specific input combination (see). For example, if input A is 0 and input B is 1, then the PRS output will assume the value of bit 1 of FNSEL (FNSEL[1]).

Table 4.1. Configurable Logic Look up Table

A	B	FNSEL
0	0	FNSEL[0]
0	1	FNSEL[1]
1	0	FNSEL[2]
1	1	FNSEL[3]

Using the FNSEL field, a total of 16 two-input logic functions can be implemented, as shown in Table 4.2 List of Logic Functions on page 14.

Table 4.2. List of Logic Functions

FNSEL value	Implemented Function
0x0	0
0x1	A NOR B
0x2	(NOT A) AND B
0x3	NOT A
0x4	A AND (NOT B)
0x5	(NOT B)
0x6	A XOR B
0x7	A NAND B
0x8	A AND B
0x9	A XNOR B
0xA	B
0xB	(NOT A) OR B
0xC	A
0xD	A OR (NOT B)
0xE	A OR B
0xF	1

The default value of FNSEL is 0xC, meaning that the input from the selected producer goes through unchanged. This feature can be used to combine multiple channels to get even more complex functions.

4.2 Event on PRS

The PRS can be used to send events to the MCU. This is very useful in combination with the Wait For Event (WFE) instruction. Using this feature, you can, for example, set up a timer to trigger an event to the MCU periodically, every time letting the MCU continue from a WFE instruction in its program. This can help in performance-critical sections where timing is known, and the goal is to wait for an event, execute some code, then wait for another event, execute some code, and so on.

Series 1

A single PRS channel can be selected for setting up events using the SEVONPRSSEL bitfield in the PRS_CTRL register. The feature is enabled by setting the SEVONPRS bitfield in the same register.

Series 2

Any asynchronous PRS channel can be selected for setting up events using PRSSEL in PRS_CONSUMER_CORE_M33RXEV.

4.3 DMA Request on PRS

Up to two independent DMA requests can be generated by the PRS.

Series 1

The PRS signals triggering the DMA requests are selected with the SOURCESEL (= 0x1 for PRS) and SIGNAL (= 0x0 for PRSREQ0 or = 0x1 for PRSREQ1) bitfields in the LDMA_CHx_REQSEL register.

The PRS channels for DMA requests are configured in the PRS_DMAREQ0 and PRS_DMAREQ1 registers. See *AN1029: Linked Direct Memory Access Controller (LDMA)* for more information. Application notes can be found on the Silicon Labs website at www.silabs.com/32bit-appnotes or in Simplicity Studio (http://www.silabs.com/simplicity).

Series 2

The PRS asynchronous channels triggering the DMA requests are selected with the PRSSEL fields in the PRS_CONSUMER_LDMAXBAR_DMAREQx registers.The requests are set whenever the selected asynchronous PRS outputs are high.

5. Software Examples for EFM32 Gecko Series 0

The software examples below are run on the EFM32 Gecko (project `prs_example_gecko` or `STKXXX_prs_example`), Tiny Gecko (project `prs_example_tg` or `STK3300_prs_example`), and Giant Gecko Starter Kits (project `prs_example_gg` or `STK3700_prs_example`), with common source file `main_prs_example.c`.

The example is selected by the menu displayed on the segment LCD: push button PB0 is used to browse the menu and push button PB1 is used to execute the selected menu item. Press push button PB0 to exit if the selected menu item is running.

5.1 TIMER Triggered ADC Conversion

This example shows how to set up ADC0 to start a single conversion every time that TIMER0 overflows. TIMER0 sends a one HFPERCLK cycle high pulse through the PRS on each overflow and the ADC does a single conversion which is displayed on the LCD.

The figure below shows a one HFPERCLK cycle pulse sent from TIMER0 to the ADC0 on an overflow. The signal triggers a single ADC conversion. The ADC consumes pulse signals which is the same signal produced by the TIMER. In this case, there is no edge detection needed, and the PRS leaves the incoming signal unchanged.

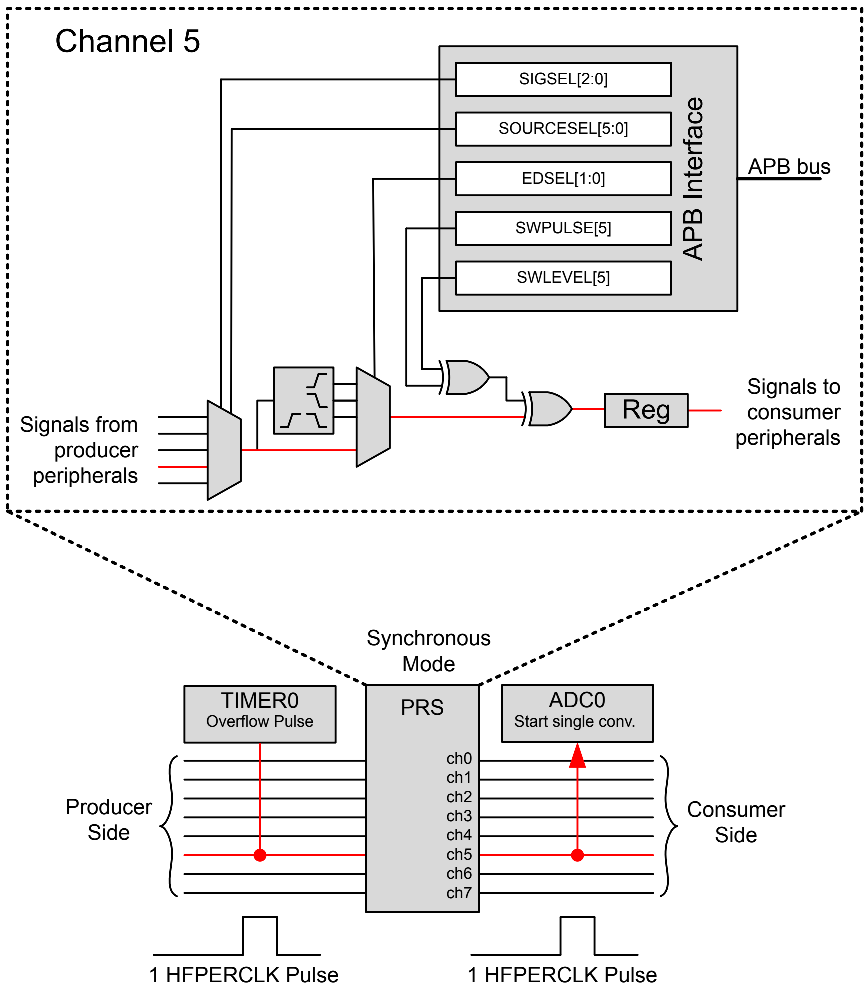


Figure 5.1. TIMER0 Overflow Starting ADC0 Single Conversions Using the PRS

The ADC is configured with 12-bit resolution and VDD as both reference and input. When the ADC finishes the conversion, it generates a single conversion complete interrupt. The MCU will then fetch the result from the ADC0_SINGLEDATA register and display it on the LCD. The DMA can also be used to fetch the conversion result, as described in the application note, *AN0021: Analog to Digital Converter (ADC)*.

The function `void prsTimerAdc (void)` in source file `prs_timer_adc.c` implements this example.

Press the push button PB0 to select the [**ADC**].

```
ADC
```

Press push button PB1 to start TIMER trigger. The TIMER triggered (approximately every one second) ADC converted voltage, for example 3.288 V for VDD, will display on the segment LCD as shown below.

```
   3288
ADC RUN
```

5.2 Pulse Width Measurement with ACMP and TIMER

This example shows how to measure the pulse width or period of an arbitrary waveform. The analog comparator (ACMP) is used to send a level signal through the PRS. TIMER0 consumes both pulse and level signals, so the PRS leaves the incoming signal unchanged. On TIMER0, the PRS signal is used as input for Compare/Capture channel 0 (CC0). TIMER0 starts counting on a rising edge and captures the counter value on a falling edge. The figure below shows the level output from the ACMP sent through the PRS to the TIMER, which measures the positive pulse width using the capture feature.

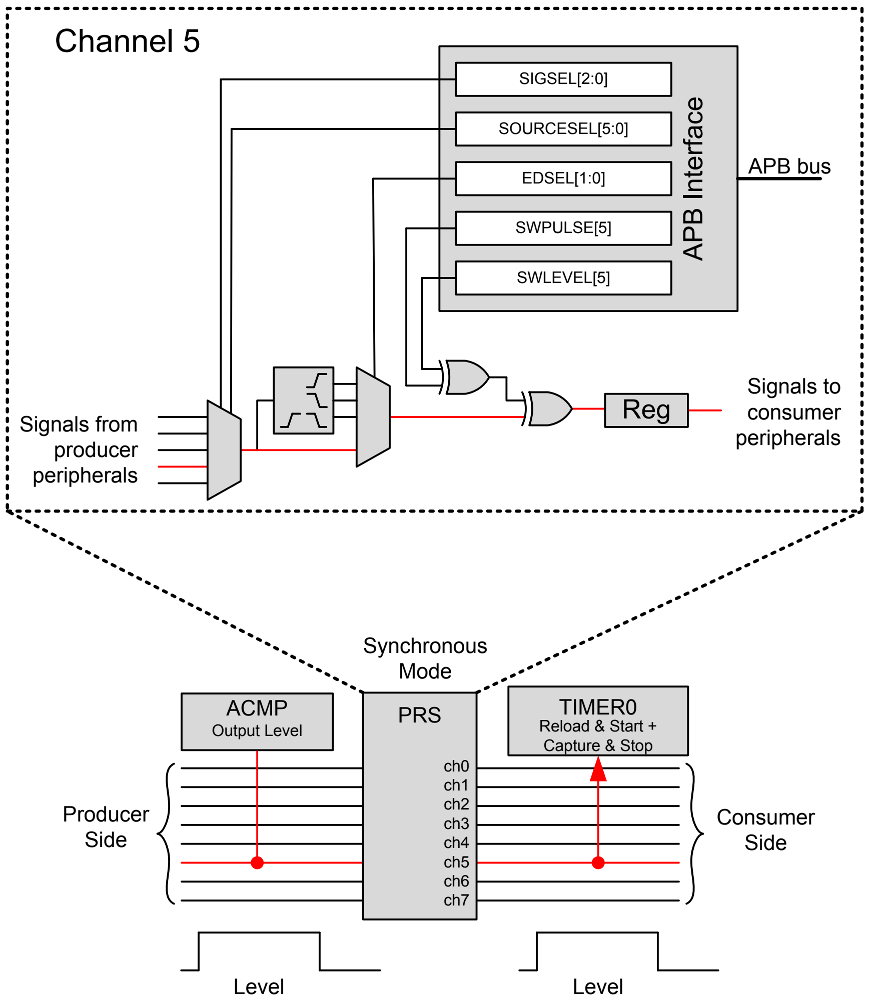


Figure 5.2. ACMP Level Output Used as PRS Signal for TIMER0 CC0 Channel Input

To trigger the pulse width measurement, pin PC4 must be connected to VMCU to generate a high level that will trigger the ACMP and start the TIMER. When the connection is released, the output of the ACMP will be low again, and the TIMER captures the counter value and displays the positive pulse width in seconds on the LCD.

The function `void prsAcmpCapture (void)` in source file `prs_pulse_width_channel_scan.c` implements this example.

Press the push button PB0 to select the [**PUL**].

```
PUL
```

Press push button PB1 to start the pulse width measurement. The positive pulse width, for example 0.578 s, will display on the segment LCD as shown below when rising then falling edges are detected on PC4 (pin 3 of EXP header on EFM32 Gecko and Tiny Gecko Starter Kit, pin 7 of EXP header on Giant Gecko Starter Kit).

```
   0578
PUL RUN
```

5.3 GPIO Triggered USART Transmission

This example shows how to use an external signal coming through the GPIO to enable the USART transmitter.

The figure below shows a falling edge from a GPIO pin sent on the producer side through the PRS edge detector to create a one HFPERCLK cycle pulse on the consumer side. The GPIO produces level signals which are not consumed by the USART, so the edge detector must be used to generate a pulse signal on a GPIO falling edge transition. The clock pulse enables the USART TX and transmits the data that was placed in the TX buffer.

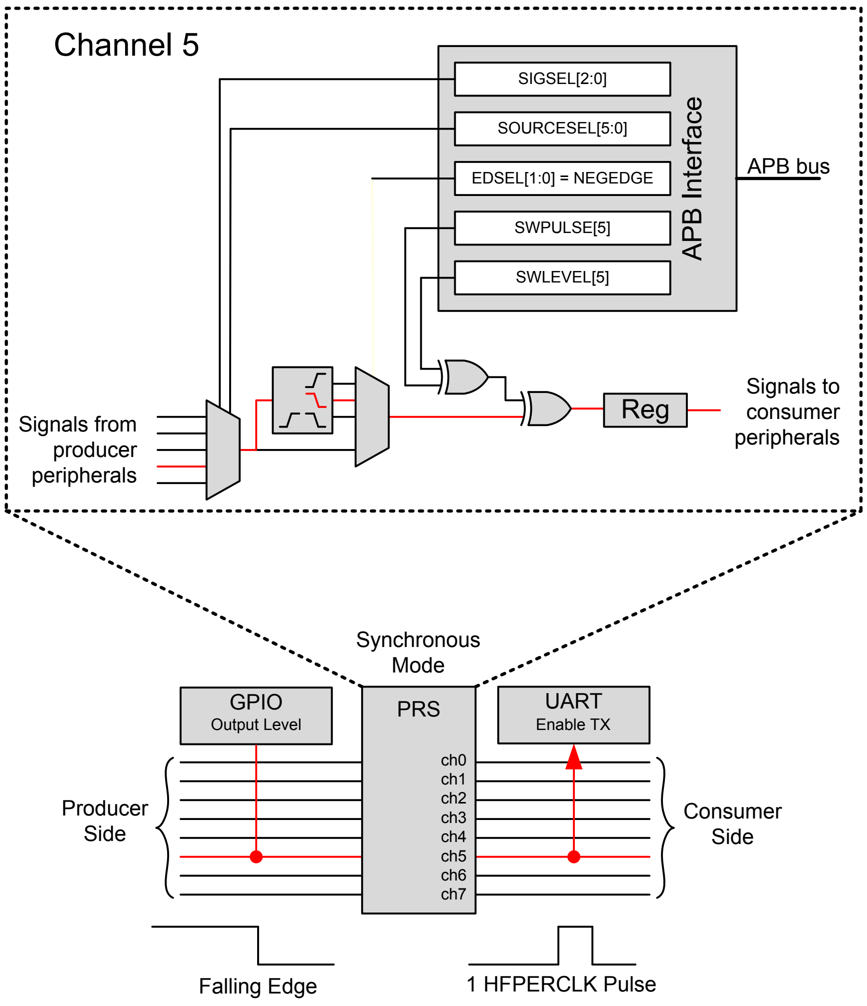


Figure 5.3. USART TX Enabled by GPIO Signal Using the PRS

The function `void prsGpioUsart (void)` in source file `prs_gpio_usart.c` implements this example.

Press the push button PB0 to select [**UART**].

```
UART
```

Press push button PB1 to wait GPIO trigger for USART transmission. Every press on PB1 will trigger a single character (cycle from character 0 to 9) USART transmission (115200 baud rate, no parity and one stop bit) on PD0 (pin 4 of EXP header on Starter Kit).

```
   0003
UARTRUN
```

5.4 Software Generated PRS Pulse Triggers DAC Conversion

This example shows how to generate a PRS pulse by software. The PRS pulse will trigger a DAC conversion which outputs a 0.5 V signal on pin PB12 (pin 13 of EXP header on the Starter Kit). It is possible to generate both pulse and level signals by software. In this case, a pulse signal is generated because it is the type of signal consumed by the DAC. DAC conversions can also be started by software in the DAC itself. This example only shows how this can be done through the PRS as well.

The figure below shows one HFPERCLK cycle pulse triggered from software. Pulse and level signals can be generated by software by writing directly to the PRS_SWPULSE and PRS_SWLEVEL registers respectively. They can also be generated using functions from the emlib.

- `void PRS_PulseTrigger(uint32_t channels)` generates pulse signals
- `void PRS_LevelSet(uint32_t level, uint32_t mask)` generates level signals

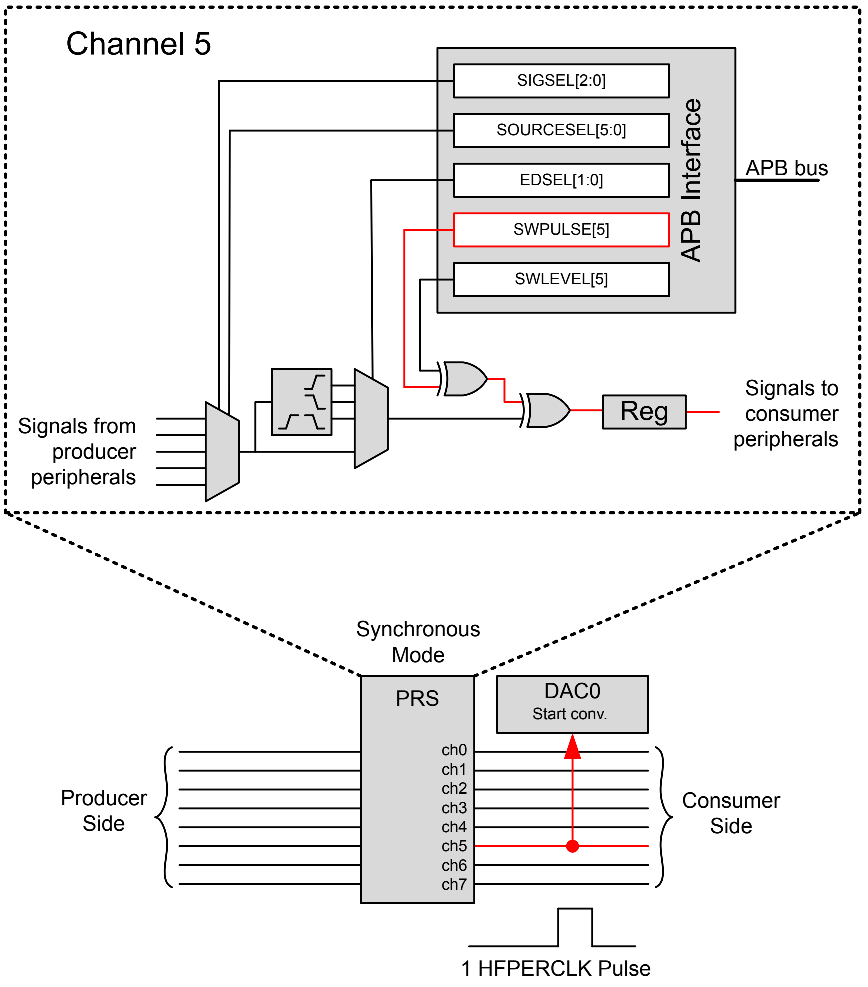


Figure 5.4. Software Triggered PRS Signal

The function `void prsSwDac (void)` in source file `prs_soft_dac.c` implements this example.

Press the push button PB0 to select [**DAC**].

```
DAC
```

Press push button PB1 to start the software trigger and the DAC voltage (0.5 V) will display on the segment LCD as shown below.

```
   0500
DAC RUN
```

5.5 Monitoring of PRS Signals

The PRS channels can be monitored using peripherals that consume PRS signals. One example is using a TIMER to make a capture when there is activity on the PRS channel it is connected to.

The Compare/Capture channel 0 (CC0) on TIMER0 is used to capture a falling edge. When a capture is triggered, the user knows that there was activity on the selected PRS channel.

This example will wait on Energy Mode 1 (EM1) for activity in the PRS channel (TIMER0 capture interrupt). When such activity occurs, it writes the PRS trigger count on the LCD. To generate activity on this line, the user must connect PC4 (pin 3 of EXP header on EFM32 Gecko and Tiny Gecko Starter Kit, pin 7 of EXP header on Giant Gecko Starter Kit) to VMCU to generate a rising edge transition on the PRS channel using the analog comparator (ACMP).

This example is useful for EFM32 Gecko (EFM32G) devices that cannot route a PRS channel to a GPIO pin for monitoring or debugging.

The function `void prsMonitor (void)` in source file `prs_pulse_width_channel_scan.c` implements this example.

Press the push button PB0 to select [**MON**].

```
MON
```

Press push button PB1 to start the monitor process and the PRS trigger count will display on the segment LCD as shown below.

```
   0003
MON RUN
```

5.6 Asynchronous GPIO PRS Triggered PCNT in EM2

This example shows how to use an external signal coming through the GPIO to clock the pulse counter (PCNT). Asynchronous mode PRS is used since both GPIO and PCNT can support this feature.

The figure below shows the level output from the GPIO sent through the asynchronous PRS to the PCNT which clocks the PCNT counter at a rising edge. The LFRCO is used for the LFA clock (LCD and PCNT) so this example can run in Energy Mode 2 (EM2) but not in Energy Mode 3 (EM3).

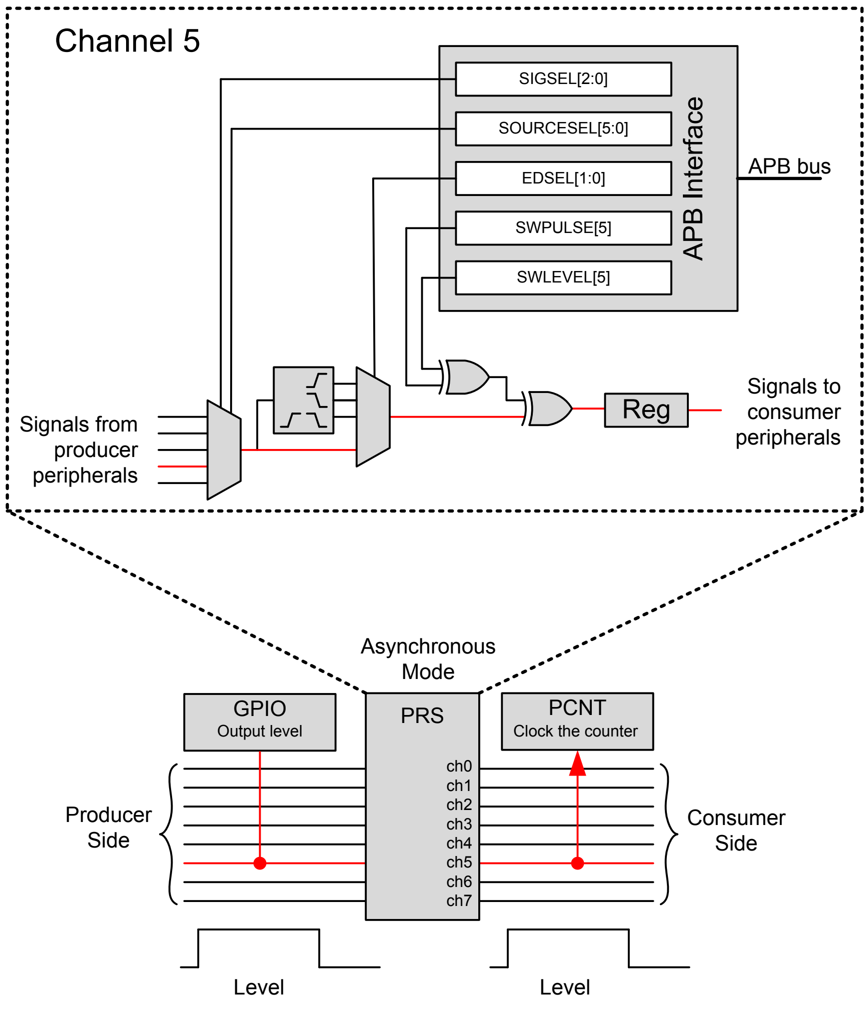


Figure 5.5. PCNT clocked by GPIO signal using the PRS

The function `void prsGpioPcnt (void)` in source file `prs_gpio_pcnt.c` implements this example.

Note: The EFM32 Gecko (EFM32G) device does not support asynchronous mode PRS, so this example is not available on the EFM32 Gecko STK.

Press the push button PB0 to select [**PCNT**].

```
PCNT
```

Press push button PB1 to trigger the PCNT counter and the counter value will display on the segment LCD as shown below.

```
   0003
PCNTRUN
```

6. Software Examples for EFM32 Gecko Series 1

The software examples below are run on the EFM32 Pearl Gecko Starter Kit (SLSTK3401A_EFM32PG). These examples are grouped into one project (`prs_example_pg` or `SLSTK3401A_prs_example`) with source file `main_prs_example.c`.

The example is selected by the menu displayed on the Memory LCD, push button BTN1 is used to browse the menu, and push button BTN0 is used to execute the selected menu item. Press push button BTN1 to exit if the selected menu item is running.

6.1 TIMER Triggered ADC Conversion

Refer to 5.1 TIMER Triggered ADC Conversion for a detailed description of this example. The function `void prsTimerAdc (void)` in source file `main_prs_example_gecko_s1.c` implements this example.

Press the push button BTN1 to select [**Timer Triggered ADC Conversion**].

```
Example 1
Timer Triggered
ADC Conversion

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to start the TIMER trigger. The TIMER-triggered (approximately every one second) ADC-converted voltage will display on the Memory LCD as shown below.

```
Example 1
Timer Triggered
ADC Conversion

Trigger interval ~1s
AVDD: 3.2887V

Press BTN1 to exit
```

6.2 Pulse Width Measurement with ACMP and TIMER

Refer to 5.2 Pulse Width Measurement with ACMP and TIMER for a detailed description of this example. The function `void prsAcmpCapture (void)` in source file `prs_pulse_width_channel_scan.c` implements this example.

Press the push button BTN1 to select [**Pulse Width Measurement on PA4 with ACMP and TIMER**].

```
Example 2
Pulse Width
Measurement on PA4
with ACMP and TIMER

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to start the pulse width measurement. The pulse width will display on the Memory LCD as shown below when rising then falling edges are detected on PA4 (pin 7 of EXP header on Starter Kit).

```
Example 2
Pulse Width
Measurement on PA4
with ACMP and TIMER

Positive pulse width:
0.2399s

Press BTN1 to exit
```

6.3 GPIO Triggered USART Transmission

Refer to 5.3 GPIO Triggered USART Transmission for a detailed description of this example. The function `void prsGpioUsart (void)` in source file `prs_gpio_usart.c` implements this example.

Press the push button BTN1 to select [**GPIO Triggered USART Transmission**].

```
Example 3
GPIO Triggered USART
Transmission

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to start GPIO triggers for USART transmission. Every press on BTN0 will trigger a single character (cycle from character 0 to 9) USART transmission (115200 baud rate, no parity and one stop bit) on PC7 (pin 6 of EXP header on Starter Kit).

```
Example 3
GPIO Triggered USART
Transmission

Press BTN0 to trigger
USART TX on PC7

Character: 1

Press BTN1 to exit
```

6.4 Asynchronous GPIO PRS Triggered PCNT in EM3

Refer to 5.6 Asynchronous GPIO PRS Triggered PCNT in EM2 for a detailed description of this example. The function `void prsGpioPcnt (void)` in source file `prs_gpio_pcnt.c` implements this example. The ULFRCO is used for the LFA clock (PCNT) so this example can run in Energy Mode 2 (EM2) and Energy Mode 3 (EM3).

Press the push button BTN1 to select [**Asynchronous GPIO PRS Triggered PCNT in EM3**].

```
Example 4
Asynchronous GPIO PRS
Triggered PCNT in EM3

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to trigger the PCNT counter, and the counter value will display on the Memory LCD as shown below.

```
Example 4
Asynchronous GPIO PRS
Triggered PCNT in EM3

Press BTN0 to trigger
PCNT

PCNT CNT: 00003

Press BTN1 to exit
```

6.5 Event on PRS

This example demonstrates how to send event to the MCU through the PRS.

The TIMER0 is setup to trigger an event to the MCU periodically (approximately every one second), every time letting the MCU pass through a Wait For Event (WFE) instruction in its program.

This can help in performance critical sections where timing is known, and the goal is to wait for an event, then execute some code, then wait for an event, then execute some code and so on.

The function `void prsTimerWfe (void)` in source file `prs_timer_wfe.c` implements this example.

Press the push button BTN1 to select [**Event on PRS - WFE in EM1**].

```
Example 5
Event on PRS - WFE in
EM1

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to start the TIMER trigger, and the wake up count will display on the Memory LCD as shown below.

```
Example 5
Event on PRS - WFE in
EM1

Periodic (~1s) PRS
Event from TIMER

PRS Wakeup: 0008

Press BTN1 to exit
```

6.6 Configurable PRS Logic

This example demonstrates how to use the configurable logic in PRS to reduce the current consumption in a thermistor application.

The target is to evaluate the ADC result (temperature) autonomously and only give the MCU an interrupt if the sample is outside or inside the given thresholds.

The functions `void rtccSetup (void)`, `void adcPrsLogic (void)`, `void adcSetup (void)`, and `void prsConfigLogic (void)` in source file `prs_rtcc_logic.c` implement this example.

6.6.1 Design Concept

Power gating is critical in this scenario, since the current consumption of the thermistor (~30 to 60 µA) becomes dominant in sleep mode (~2 µA) unless the MCU makes sure the thermistor is powered only when it is being sampled.

This can be achieved by routing the RTCC signal (RTCC Event) to a GPIO pin to drive the thermistor. This scenario is shown in Figure 6.1 ADC Sample Triggered by RTCC Event through PRS on page 26.

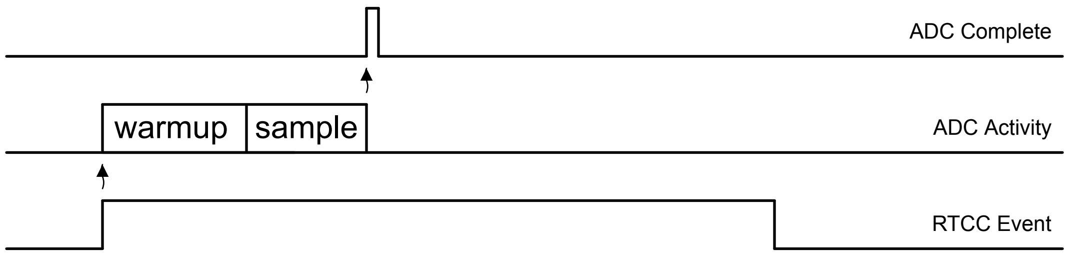


Figure 6.1. ADC Sample Triggered by RTCC Event through PRS

The RTCC Event pulse width is ~30.5 µs if based on a 32768 Hz clock (LFRCO or LFXO) and 1 ms if based on a 1 kHz clock (ULFRCO). As the figure above shows, the RTCC Event is longer than the ADC takes to warm up and sample the thermistor, so the RTCC Event keeps the thermistor on longer than necessary.

The thermistor on time can be further reduced by using combinational logic. For example, the ADC produces a short PRS output whenever it is done. Using the RTCC event PRS signal and the ADC conversion done PRS pulse, it is possible to create a signal that goes high on the RTC trigger and low when the ADC is complete. This signal can automatically enable the thermistor in the system.

The conceptual circuit for this example, which starts external sensor excitation on RTCC trigger and ends excitation when ADC sample has been taken, is shown in Figure 6.2 Conceptual Circuit on page 26 . The truth table of this circuit is shown in Table 6.1 Truth Table of Conceptual Circuit on page 27.

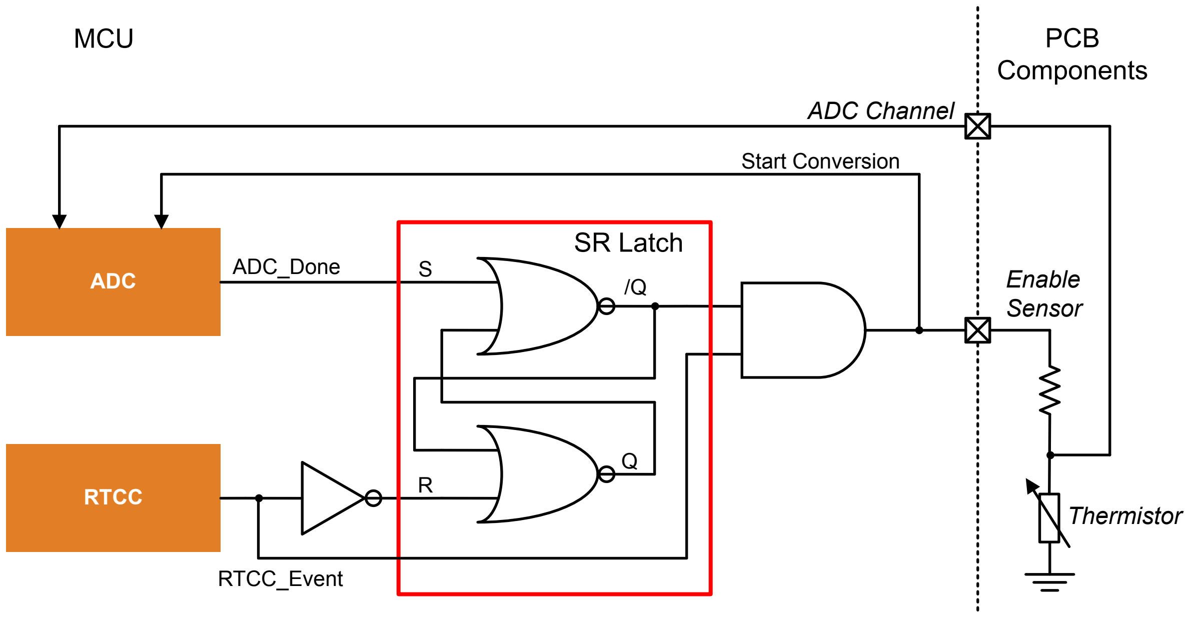


Figure 6.2. Conceptual Circuit

Table 6.1. Truth Table of Conceptual Circuit

RTCC_Event	SR Latch Input S (ADC_Done)	SR Latch Input R (/RTCC_Event)	SR Latch Output /Q	Enable Sensor (RTCC_Event & /Q)
0	0	1	1	0
1	0	0	1	1
1	1	0	0	0
1	0	0	0	0

Initially, the SR latch output /Q is high and Enable Sensor is low because the RTCC_Event output and the ADC_Done output are low. Whenever the RTCC_Event now goes high, the external sensor is enabled, and the ADC starts taking a sample. Once the ADC is done, the ADC_Done signal goes high, setting the SR latch output /Q low, which forces the external sensor off. When the RTCC event signal goes low again, the SR latch is reset, making the system ready for the next event.

6.6.2 Design Implementation

The circuit shown in Figure 6.2 Conceptual Circuit on page 26 can easily be implemented on an EFM32 Pearl Gecko (EFM32PG) using the PRS, which contains logic elements that can be connected together in various ways.

The Figure 6.3 PRS Logic Implementation for Excitation of External Sensor on page 29 shows what this implementation would look like using six of the available PRS channels. The Table 6.2 PRS Logic Configuration for Excitation of External Sensor on page 30 shows how the six PRS channels are configured to enable this functionality.

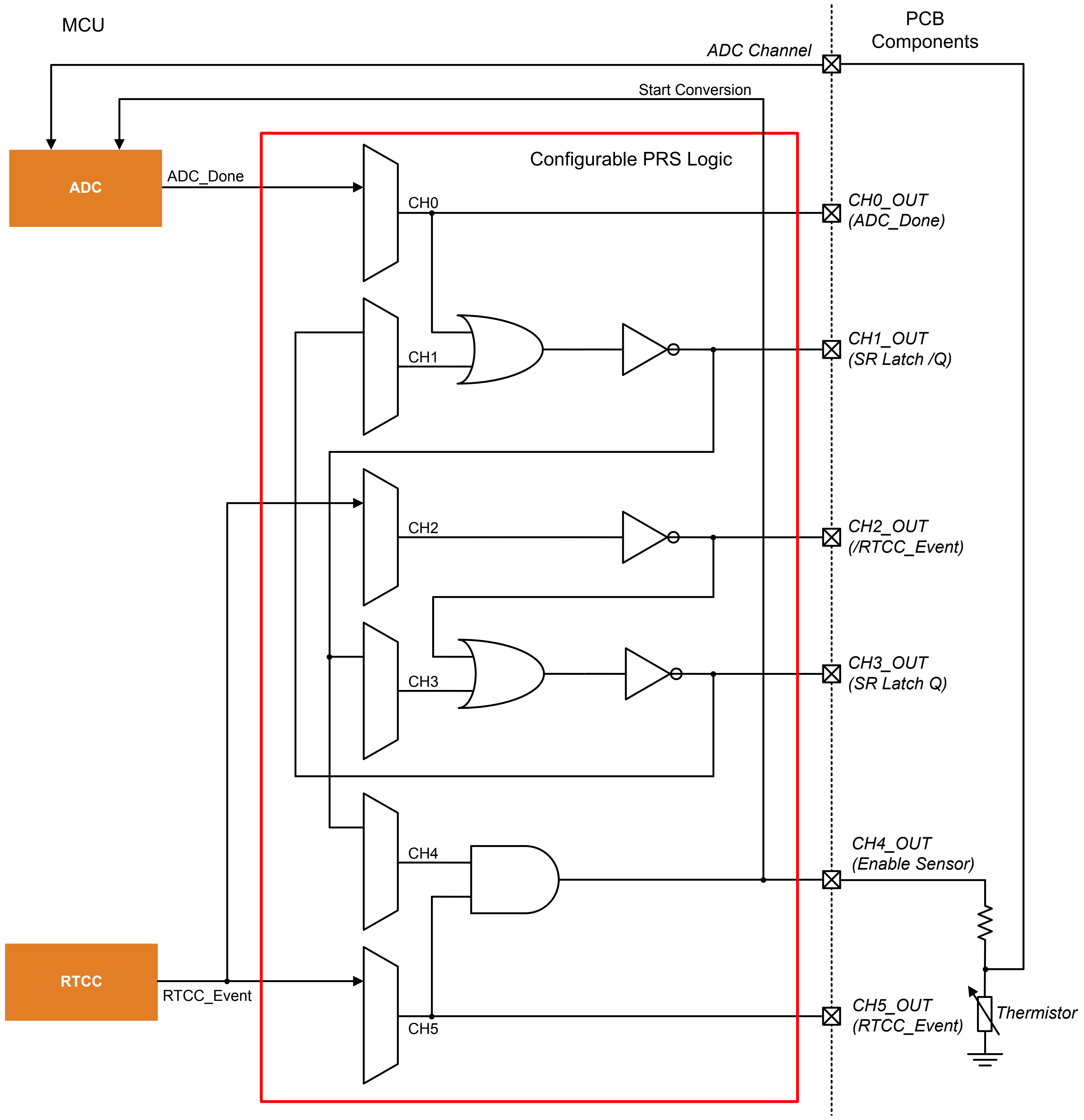


Figure 6.3. PRS Logic Implementation for Excitation of External Sensor

Table 6.2. PRS Logic Configuration for Excitation of External Sensor

PRS Channel	Input	ANDNEXT	ORPREV	INV	Output
0	ADC_Done	—	—	—	ADC_Done
1	PRS_CH3	—	PRS_CH0	Y	/(PRS_CH3 | PRS_CH0)
2	RTCC_Event	—	—	Y	/RTCC_Event
3	PRS_CH1	—	PRS_CH2	Y	/(PRS_CH1 | PRS_CH2)
4	PRS_CH1	PRS_CH5	—	—	PRS_CH1 & PRS_CH5 (Enable Sensor)
5	RTCC_Event	—	—	—	RTCC_Event

The configuration of Table 6.2 PRS Logic Configuration for Excitation of External Sensor on page 30 can be realized by C source code as shown below.

```
CMU_ClockEnable(cmuClock_PRS, true);

/* Use ADC SINGLE as an ASYNC PRS producer on CH0 */
PRS_SourceAsyncSignalSet(0, PRS_CH_CTRL_SOURCESEL_ADC0, PRS_CH_CTRL_SIGSEL_ADC0SINGLE);

/* CH3 OR CH0 then INV on CH1 */
PRS_SourceAsyncSignalSet(1, PRS_CH_CTRL_SOURCESEL_PRSL, PRS_CH_CTRL_SIGSEL_PRSCH3);
PRS->CH[1].CTRL |= PRS_CH_CTRL_ORPREV | PRS_CH_CTRL_INV;

/* Invert RTCC trigger on CH2 */
PRS_SourceAsyncSignalSet(2, PRS_CH_CTRL_SOURCESEL_RTCC, PRS_CH_CTRL_SIGSEL_RTCCCCV1);
PRS->CH[2].CTRL |= PRS_CH_CTRL_INV;

/* CH1 OR CH2 then INV on CH3 */
PRS_SourceAsyncSignalSet(3, PRS_CH_CTRL_SOURCESEL_PRSL, PRS_CH_CTRL_SIGSEL_PRSCH1);
PRS->CH[3].CTRL |= PRS_CH_CTRL_ORPREV | PRS_CH_CTRL_INV;

/* CH1 AND CH5 on CH4 */
PRS_SourceAsyncSignalSet(4, PRS_CH_CTRL_SOURCESEL_PRSL, PRS_CH_CTRL_SIGSEL_PRSCH1);
PRS->CH[4].CTRL |= PRS_CH_CTRL_ANDNEXT;

/* Use RTCC as an ASYNC PRS producer on CH5 */
PRS_SourceAsyncSignalSet(5, PRS_CH_CTRL_SOURCESEL_RTCC, PRS_CH_CTRL_SIGSEL_RTCCCCV1);
```

6.6.3 Test Results

This example uses the configurable PRS logic to trigger the ADC single conversion at 100 Hz. The EFM32 Pearl Gecko stays in Energy Mode 3 (EM3) and wakes up with the ADC SINGLECMP interrupt.

The clock configurations for this example are listed below.

- Core clock — 16 MHz HFRCO
- ADC conversion clock — 13 MHz AUXHFRCO in ASYNC mode
- RTCC — Use 1 kHz ULFRCO as clock source for EM3 operation

The ADC configurations for this example are listed below.

- PRS enable — Trigger from PRS channel 4
- Reference — AVDD
- Input — PA0 (pin 12 of EXP header on EFM32PG Starter Kit)
- FIFO overflow action — FIFO overwrites old data when full
- ADC acquisition time — One ADC conversion clock cycle
- ADC conversion resolution — 12 bit
- ADC bias programming — Set GPBIASACC bitfield in the ADCn_BIASPROG register to LOWACC
- ADC clock mode — ASYNC and use AUXHFRCO as clock source for EM3 operation
- ADC interrupt — Single result compare matched (SINGLECMP)

The ADC compare thresholds below are used in this example.

- ADC ADGT — 3724 (~3V for AVDD = 3.3 V)
- ADC ADLT — 620 (~0.5V for AVDD = 3.3 V)

The ADC compare matched interrupt will be triggered when PA0 input voltage is lower than the ADLT threshold or higher than the ADGT threshold.

Two 100 kΩ resistors are used to emulate the thermistor potential divider, and the current consumption is about 16 μA (3.3 V/200000) when the Enable Sensor signal is high.

The PRS channel signal can route to a GPIO pin for debugging or to use it as an enable signal. Table 6.3 GPIO Pins Used for PRS Channels on page 31 shows which GPIO pins are used in this example for different PRS channels.

Except GPIO for PRS channel 4 (Sensor Enable), all PRS channel outputs are disabled by default to reduce the current consumption. Set the `PRS_DEBUG_OUT` define in `prs_rtcc_logic.c` to 1 to enable all PRS channel outputs.

Table 6.3. GPIO Pins Used for PRS Channels

PRS Channel	Signal	GPIO	I/O Routing Location	Pin on EFM32PG STK
0	ADC_Done	PF2	2	Pin 9 of J102
1	SR Latch /Q	PF3	2	Pin 11 of J102
2	/RTCC_Event	PF4 (share with LED0)	2	Pin 13 of J102
3	SR Latch Q	PF5 (share with LED1)	2	Pin 15 of J102
4	Enable Sensor	PD10	1	• Pin 8 of J102 • Pin 9 of EXP Header
5	RTCC_Event	PD11	1	• Pin 10 of J102 • Pin 11 of EXP Header

The timing diagram of the PRS channels is shown in the figure below. The RTCC trigger pulse width is about 1 ms (ULFRCO), and the ADC sampling frequency is about 100 Hz.

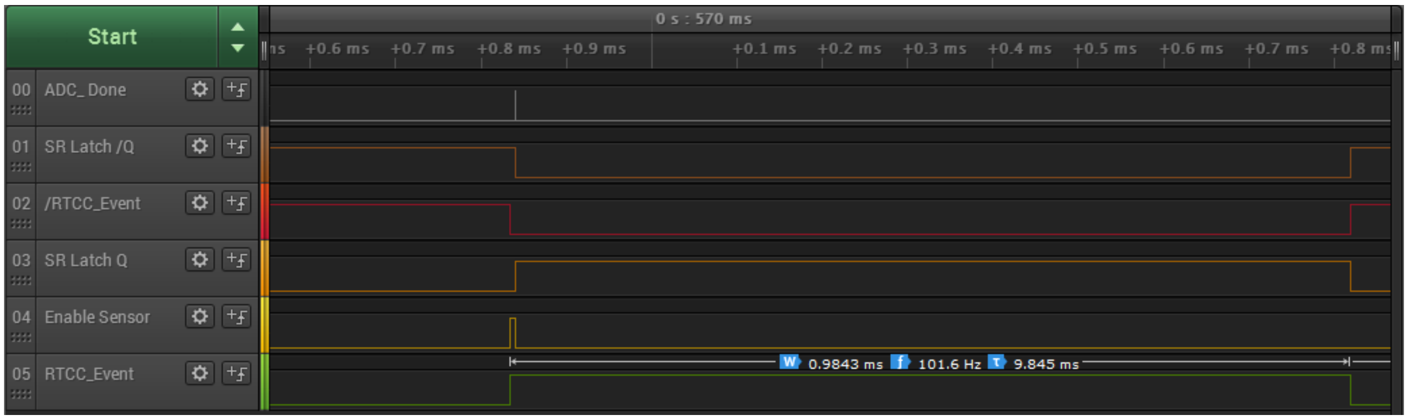


Figure 6.4. Timing Diagram of PRS Channels

The Figure 6.5 Timing Diagram of PRS Channel 0, 4, and 5 on page 32 matches with the diagram in Figure 6.1 ADC Sample Triggered by RTCC Event through PRS on page 26. The Enable Sensor pulse width includes the delay to start the 13 MHz AUXHFRCO oscillator, and the ADC warmup and sample time is 6.875 µs.

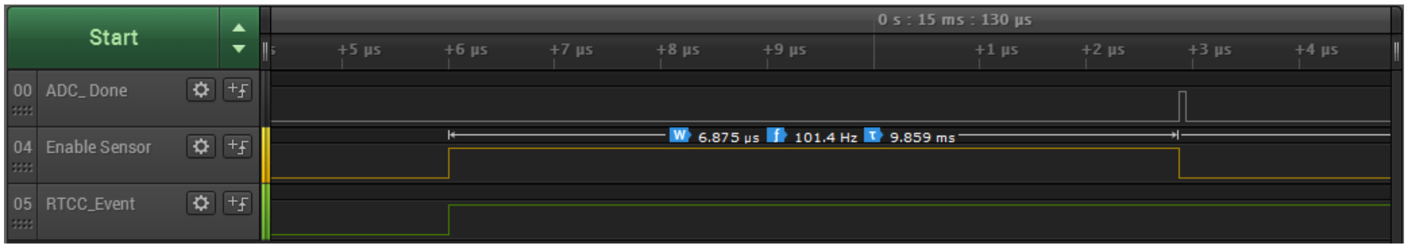


Figure 6.5. Timing Diagram of PRS Channel 0, 4, and 5

Press the push button BTN1 to select [**RTCC Triggered ADC Conversion with Configurable PRS Logic in EM3**].

```
Example 5
RTCC Triggered
ADC Conversion
with Configurable
PRS Logic in EM3

Press BTN1 to next
menu
Press BTN0 to start
```

Press push button BTN0 to place the EFM32 Pearl Gecko into Energy Mode 3 (EM3).

```
Example 5
RTCC Triggered
ADC Conversion
with Configurable
PRS Logic in EM3

Sleep in EM3

Wakeup if voltage in
PA0 is outside the
compare window
(0.5V – 3V)

Press BTN1 to exit
```

The MCU wakes up when the ADC SINGCMP interrupt triggers, and voltages in the latest single FIFO display on the Memory LCD as shown below.

```
Example 5
RTCC Triggered
```

```
ADC Conversion
with Configurable
PRS Logic in EM3

ADC Compare Interrupt

FIFO 0: 1.4212V
FIFO 1: 1.4212V
FIFO 2: 1.4212V
FIFO 3: 0.0008V

Press BTN1 to exit
```

To measure the current consumption in Energy Mode 3 (EM3), the debugger must be disconnected from the IDE. Then, switch the power selector on the EFM32 Pearl Gecko Starter Kit to the "BAT" position and then back to the "AEM" position to provide a Power-on Reset (POR) to the DC-DC converter.

The average current consumption of this example is about 2.77 µA (measured by Energy Profiler in Simplicity Studio), and this figure includes the power consumption of the inactive Memory LCD on the EFM32PG STK.

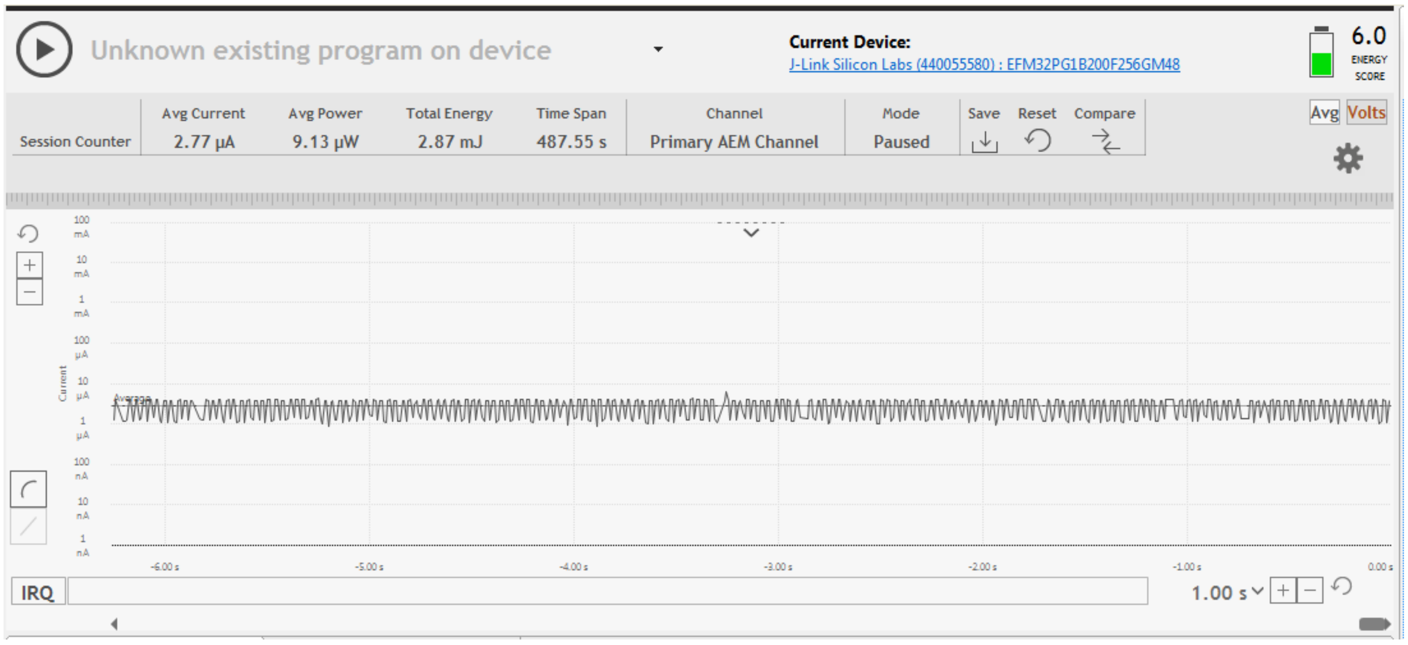


Figure 6.6. Current Consumption of Configurable PRS Logic Example

7. Software Examples for EFR32 Gecko Series 2

Software examples for the PRS on EFR32 Gecko Series 2 can be found in the Silicon Labs Peripheral Example repository on Github - https://github.com/SiliconLabs/peripheral_examples

7.1 TIMER Triggered IADC Conversion

The `<kit>_iadc_scan_letimer_prs_ldma` and `<kit>_iadc_single_letimer_prs_ldma` examples in Github repository demonstrate timer-triggered IADC conversions.

`<kit>_iadc_scan_letimer_prs_ldma`: This example utilizes the IADC to perform repeated, non-blocking measurements on two external inputs. The LDMA then transfers the measured values to memory, all while operating in EM2. The LETIMER, running in EM2, periodically requests IADC conversions via PRS. After `NUM_SAMPLES` conversions, the LDMA triggers an interrupt from EM2 and toggles LED0 on the WSTK.

`<kit>_iadc_single_letimer_prs_ldma`:

This example showcases repeated, non-blocking IADC measurements on a single channel. The LETIMER initiates conversions via PRS, while the LDMA transfers results to RAM, maintaining operation in EM2. The LETIMER trigger pulse is observable on a GPIO, while the completion of each LDMA transfer sequence drives LED0 on the Wireless Starter Kit mainboard.

7.2 GPIO Triggered IADC Conversion

The `<kit>_iadc_scan_gpio_prs_ldma` and `<kit>_iadc_single_gpio_prs_ldma` examples in Github repository demonstrate GPIO-triggered IADC conversions.

`<kit>_iadc_scan_gpio_prs_ldma`: This example utilizes the IADC to perform repeated, non-blocking measurements on two external inputs. Conversions are initiated via PRS in response to a rising edge on a user-defined GPIO pin, with the LDMA transferring the results to RAM, all while remaining in EM2. The IADC records the results and requests an interrupt upon completion of all transfers. After collecting the specified number of samples, a GPIO output is toggled on the Wireless Starter Kit mainboard.

`<kit>_iadc_single_gpio_prs_ldma`: This example showcases repeated non-blocking IADC measurements on a single input. A GPIO rising edge triggers conversions via PRS, and the LDMA transfers the measured data to memory, maintaining operation in EM2. After NUM_SAMPLES conversions, the LDMA triggers an interrupt from EM2 and toggles LED0 on the Wireless Starter Kit mainboard.

7.3 GPIO Triggered PCNT Event

The `<kit>_pcnt_extclk_quadrature`, `<kit_pcnt_extclk_single_overflow`, `<kit>_pcnt_extclk_single_underflow`, `<kit>_pcnt_oversampling_quadrature`, `<kit>_pcnt_single_oversampling_overflow`, and `<kit>_pcnt_single_oversampling_underflow` examples in Github repository illustrate how GPIO can be used to trigger the PCNT peripheral.

`<kit>_pcnt_extclk_quadrature`: This example demonstrates use of the Pulse Counter (PCNT) peripheral in external quadrature mode, utilizing PRS for efficient signal routing. Push buttons (PB0 and PB1) are configured as PRS sources, allowing PCNT0 to track direction changes from an external quadrature encoder. The PCNT peripheral processes quadrature signals via PRS channels, enabling hardware-based counting without CPU intervention. When a direction change occurs, an interrupt toggles LED0 or LED1 accordingly.

`<kit>_pcnt_oversampling_quadrature`: This example demonstrates oversampling quadrature decoder mode of the PCNT peripheral, using PRS for quadrature signal processing. The project configures S0IN and S1IN as PRS inputs, implementing OVSQUAD 4X mode for high-resolution quadrature decoding. The counter updates with each state transition, triggering interrupts on direction changes, counter overflow, or underflow. LED0 toggles for direction changes, while LED1 toggles in response to overflow or underflow events.

`<kit_pcnt_extclk_single_overflow`: This example demonstrates externally clocked single input counter mode of the PCNT peripheral, employing PRS for event-driven pulse counting. Push Button PB0 functions as both the pulse counter input and external clock source, generating a clock cycle upon each press and release. The PCNT counter starts at zero and increments with each press. Upon reaching the predefined top value, an interrupt is triggered, causing LED0 to toggle, indicating an overflow event.

`<kit>_pcnt_extclk_single_underflow`: This example demonstrates externally clocked single input counting using PRS. Push Button PB0 acts as both the counter input and external clock source. The PCNT counter starts from a predefined top value and decrements with each button press. Once the counter reaches zero, an interrupt toggles LED0 to signal an underflow event.

`<kit>_pcnt_single_oversampling_overflow`: This example demonstrates externally clocked single input counting with PRS-driven pulse counting. PB0 serves as the pulse counter input and clock source, generating a clock cycle per press and release. The counter starts at zero, incrementing with each press until it reaches the configured top value, at which point an interrupt toggles LED0 to indicate an overflow.

`<kit>_pcnt_single_oversampling_underflow`: This example demonstrates externally clocked single input counting with PRS. PB0 functions as both the pulse counter input and external clock source. The PCNT counter begins from a predefined top value, decrementing with each button press. Upon reaching zero, an interrupt toggles LED0, signaling an underflow event.

7.4 GPIO Triggered LDMA Transfer

The `<kit>_ldma_single_direct_register_transfer` and `<kit>_ldma_inter_channel_sync` examples in the GitHub repository showcase the functionality of LDMA controller, utilizing the PRS for event-driven DMA operations.

`<kit>_ldma_single_direct_register_transfer`: This example demonstrates a single direct register LDMA transfer, leveraging PRS for efficient event-driven data movement. This project configures an LDMA channel to execute memory-to-memory transfers with an optional GPIO-based PRS trigger. Push Button PB1 can act as a PRS source, initiating an LDMA transfer upon signal detection. The system operates autonomously, enabling optimized performance by facilitating data transfers without direct CPU intervention.

`<kit>_ldma_inter_channel_sync`: This example demonstrates inter-channel synchronization within the LDMA controller, using PRS for precise coordination of LDMA transfers. The project sets up two LDMA channels to handle memory-to-memory transfers, synchronizing their operations through PRS signals and hardware event triggers. Push Buttons PB0 and PB1 function as PRS sources, managing LDMA requests. Channel 0 holds execution until synchronization is achieved, while Channel 1 completes its operation and sets the synchronization flag.

7.5 Configurable PRS Logic

The `<kit>_prs_logic_unit` example in Github repository showcases the built-in PRS logic functions between channels.

It demonstrates the built-in PRS logic operates between channels by setting onboard push-buttons (PB0 and PB1) as PRS sources and configuring the onboard LED (LED1) as the PRS output.

8. Revision History

8.1 Revision 1.1

June, 2025

- Added 1. Device Compatibility section.
- Updated the document for Series 2 devices.

8.2 Revision 1.08

2016-11-18

Updated example code for EFM32 Gecko Series 0

Added example code for EFM32 Gecko Series 1

Updated document for EFM32 Gecko Series 1

8.3 Revision 1.07

2014-05-07

Updated example code to CMSIS 3.20.5

Changed to Silicon Labs license on code examples

Added project files for Simplicity IDE

Removed makefiles for Sourcery CodeBench Lite

8.4 Revision 1.06

2013-10-14

New cover layout

8.5 Revision 1.05

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

8.6 Revision 1.04

2012-11-12

Adapted software projects to new kit-driver and bsp structure.

Added software support for Tiny and Giant Gecko STK.

8.7 Revision 1.03

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

8.8 Revision 1.02

2011-10-21

Updated IDE project paths with new kits directory.

8.9 Revision 1.01

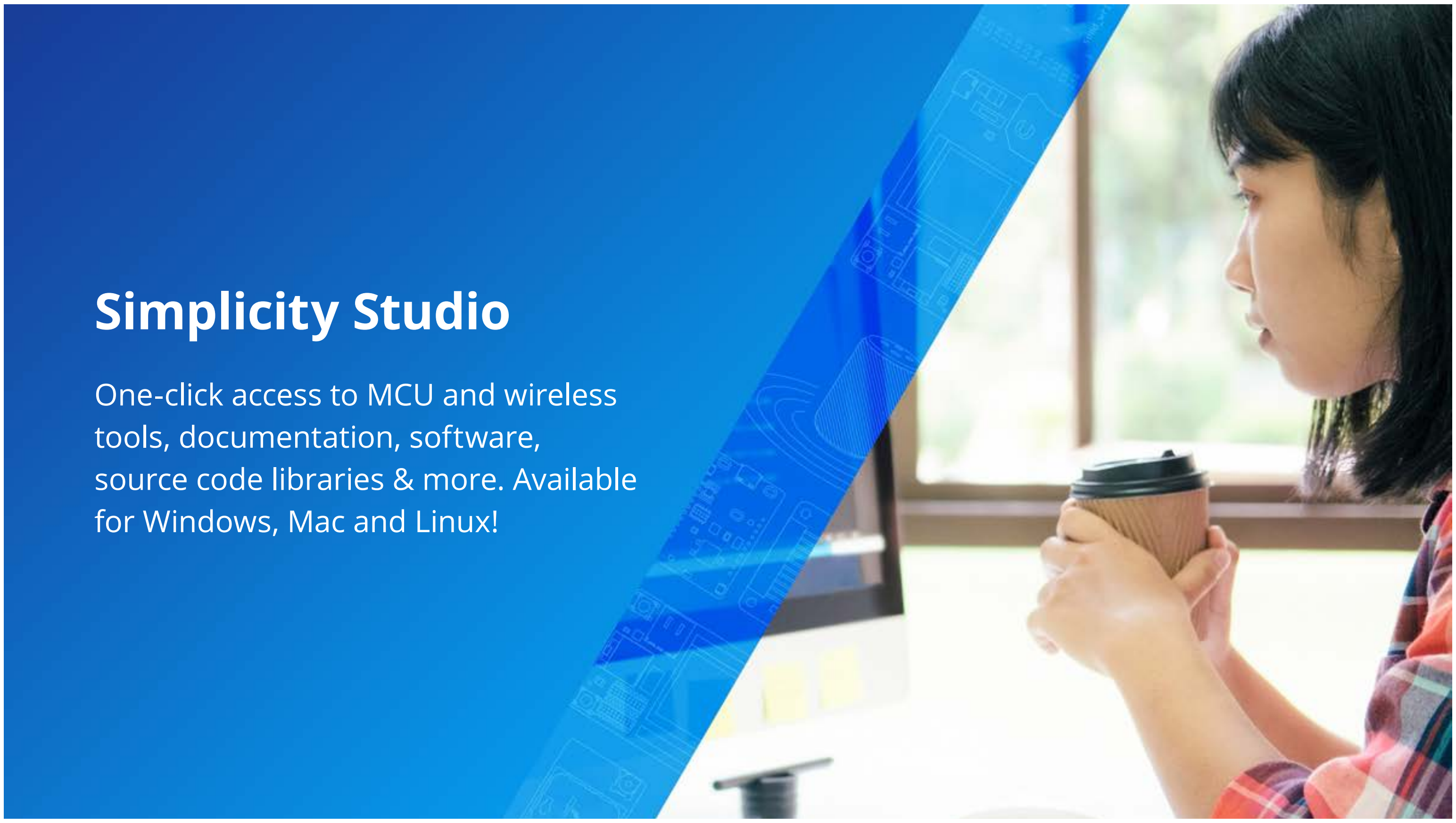
2011-05-18

Updated projects to align with new bsp version

8.10 Revision 1.00

2010-12-13

Initial revision.


Simplicity Studio
One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com