... the world's most energy friendly microcontrollers

Peripheral Reflex System

AN0025 - Application Note

Introduction

This application note describes the Peripheral Reflex System features and how these can be used to improve your system's energy performance, reduce CPU workload and latency. A few examples are presented and described in detail.

This application note includes:

- This PDF document
- Source files (zip)
  - Example C-code
  - Multiple IDE projects

1 Peripheral Reflex System

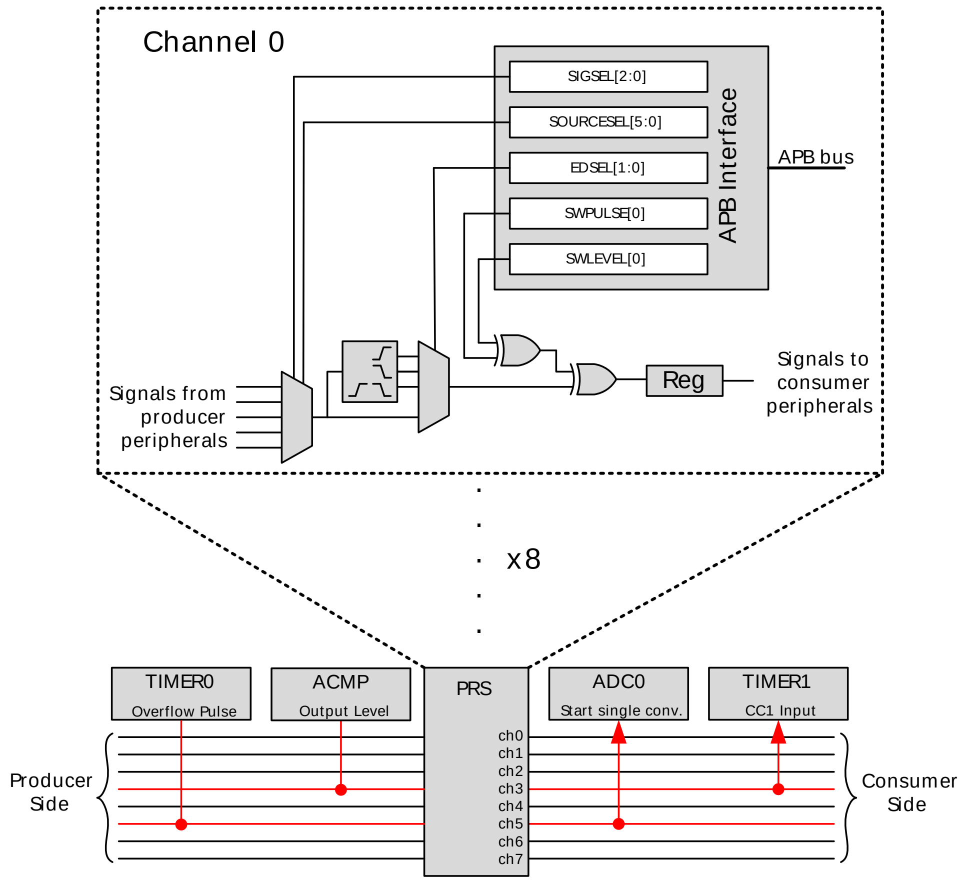
1.1 Introduction

The Peripheral Reflex System (PRS) system is a network which lets the different peripheral modules communicate directly with each other without involving the CPU. Peripheral modules which send out Reflex signals are called producers. The PRS routes these reflex signals to consumer peripherals which apply actions depending on the Reflex signals received. The format for the Reflex signals is not given, but edge triggers and other functionality can be applied by the PRS.

1.2 Overview

An overview of one channel and how 4 different peripherals can be connected PRS is given in Figure 1.1 (p. 2). The PRS contains 8 interconnect 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. Synchronous PRS pulses are one HFPERCLK cycle long, and can either be sent out by a producer (e.g. ADC conversion complete) or be generated from the edge detector in the PRS channel. Level signals can have an arbitrary waveform, but will be synchronized with the HFPERCLK.

Figure 1.1. PRS Overview



The example in Figure 1.1 (p. 2) shows 4 peripherals connected to 2 PRS channels. On one channel is TIMER0 and ADC0 and the ACMP and TIMER1 are connected to a second channel. An overflow from TIMER0 can start an ADC single conversion and an ACMP output can be used as input for a Compare/ Capture channel on TIMER1.

2 General Operation

2.1 Channel Functions

Different functions can be applied to a reflex signal within the PRS. Each channel includes an edge detector to enable generation of pulse signals from level signals. It is also possible to generate output Reflex signals by software writing to PRS_SWPULSE and PRS_SWLEVEL registers. PRS_SWLEVEL is a programmable level for each channel and holds the value it is programmed to. The PRS_SWPULSE will give out a one-cycle high pulse if it is written to 1, otherwise a 0 is asserted. The SWLEVEL and SWPULSE signals are then XOR'ed with the selected input from the producers to form the output signal sent to the consumers listening to the channel. This is illustrated in Figure 1.1 (p. 2) .

The emlib function `void PRS_SourceSignalSet(unsigned int ch, uint32_t source, uint32_t signal, PRS_Edge_TypeDef edge)` 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 the function configures the PRS accordingly.

2.2 Producers

Each PRS channel can choose between signals from several producers, which is configured in SOURCESEL in PRS_CHx_CTRL. Each of these producers outputs one or more signals which can be selected by setting the SIGSEL field in PRS_CHx_CTRL. Setting the SOURCESEL bits to 0 (Off) leads to a constant 0 output from the input mux. An overview of the available producers is given in Table 2.1 (p. 3) .

Table 2.1. Reflex Producers

Module	Reflex Output	Output Format
ACMP	Comparator Output	Level
ADC	Single Conversion Done	Pulse
	Scan Conversion Done	Pulse
DAC	Channel 0 Conversion Done	Pulse
	Channel 0 Conversion Done	Pulse
GPIO	Pin 0 Input	Level
	Pin 1 Input	Level
	Pin 2 Input	Level
	Pin 3 Input	Level
	Pin 4 Input	Level
	Pin 5 Input	Level
	Pin 6 Input	Level
	Pin 7 Input	Level
	Pin 8 Input	Level
	Pin 9 Input	Level
	Pin 10 Input	Level
	Pin 11 Input	Level
	Pin 12 Input	Level

Module	Reflex Output	Output Format
	Pin 13 Input	Level
	Pin 14 Input	Level
	Pin 15 Input	Level
RTC	Overflow	Pulse
	Compare Match 0	Pulse
	Compare Match 1	Pulse
TIMER	Underflow	Pulse
	Overflow	Pulse
	CC0 Output	Level
	CC1 Output	Level
	CC2 Output	Level
UART	TX Complete	Pulse
	RX Data Received	Pulse
USART	TX Complete	Pulse
	RX Data Received	Pulse
	IrDA Decoder Output	Level
VCMP	Comparator Output	Level

2.3 Consumers

Consumer peripherals (Listed in Table 2.2 (p. 4)) can be set to listen to a PRS channel and perform an action based on the signal received on that channel. Most consumers expect pulse input, while some can handle level inputs as well.

Table 2.2. Reflex Consumers

Module	Reflex Input	Input Format
ADC	Single Mode Trigger	Pulse
	Scan Mode Trigger	Pulse
DAC	Channel 0 Trigger	Pulse
	Channel 1 Trigger	Pulse
TIMER	CC0 Input	Pulse/Level
	CC1 Input	Pulse/Level
	CC2 Input	Pulse/Level
	DTI Fault Source 0 (TIMER0 only)	Pulse
	DTI Fault Source 1 (TIMER0 only)	Pulse
	DTI Input (TIMER0 only)	Pulse/Level
UART	TX/RX Enable	Pulse
USART	TX/RX Enable	Pulse

Module	Reflex Input	Input Format
	IrDA Encoder Input (USART0 only)	Level

3 Software Examples

This chapter describes four software examples that explore possible interactions between peripherals using the PRS:

- TIMER triggered ADC conversion
- Pulse Width Measurement with the ACMP and TIMER
- GPIO triggered UART transmission
- Software triggered DAC conversion

3.1 TIMER Triggered ADC Conversion

This example (illustrated in Figure 3.1 (p. 6)) 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.

Figure 3.1. TIMER0 overflow starting ADC0 single conversions using the PRS

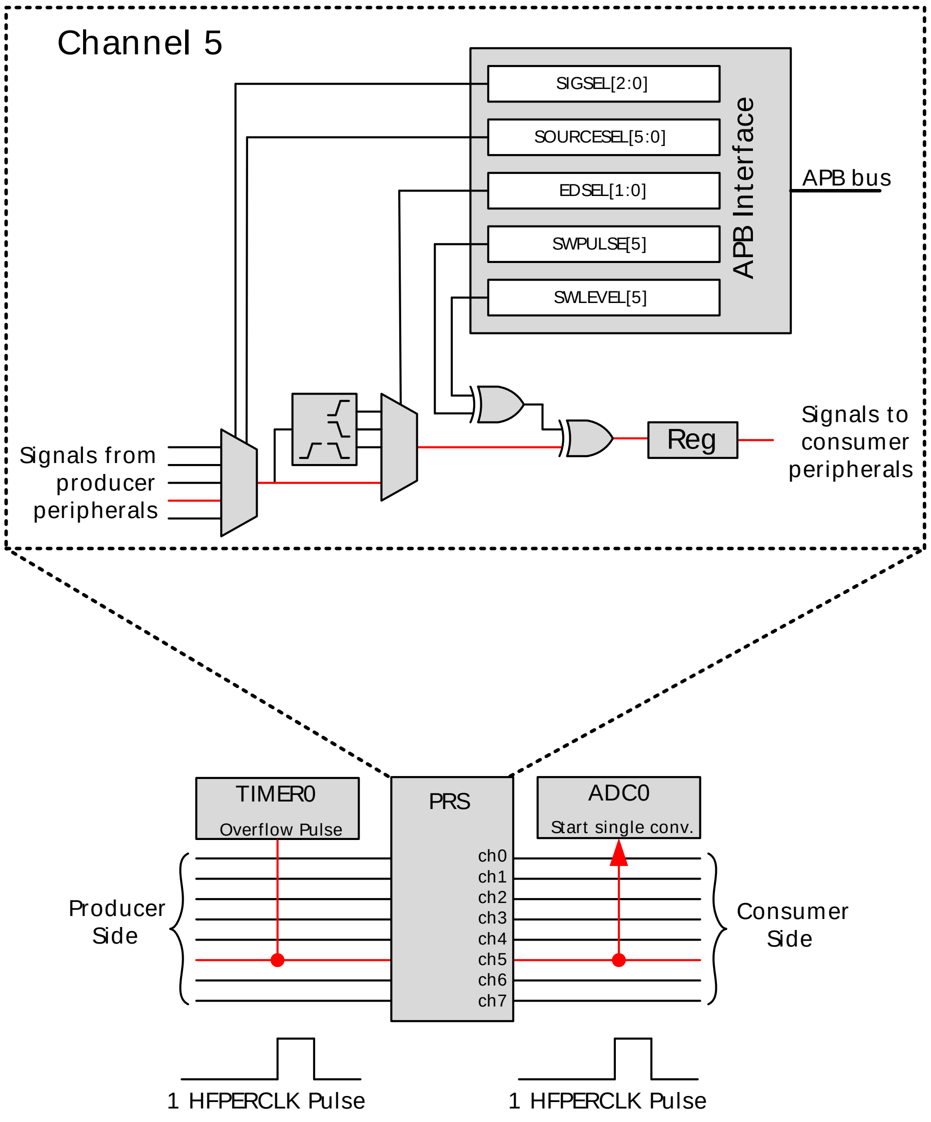


Figure 3.1 (p. 6) shows a one HFPERCLK cycle pulse sent from TIMER0 to the ADC0 on an overflow. The signal triggers a single ADC conversion. The ADC consumes pulse signals which is the

same signal produced by the TIMER. In this case there is no edge detection needed and the PRS leaves the incoming signal unchanged.

The ADC is configured with 8 bit resolution and Vdd as both reference and input. When the ADC finishes the conversion it generates a single conversion complete interrupt. The CPU will then fetch the result and display it on the LCD. The displayed result is a direct reading from the ADC0_SINGLEDATA register which is always 255 given that the input is the same as the reference. The DMA can also be used to fetch the conversion result and that is covered by AN0021 Analog to Digital Converter.

The software project `prs_timer_adc` implements this example and can be used on both STKs and DK.

3.2 Pulse Width Measurement with ACMP and TIMER

In this example (illustrated in Figure 3.2 (p. 7)) it is shown how to measure the pulse width or period of an arbitrary waveform. The 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 CC0 channel. TIMER0 starts counting on a positive edge and captures the counter value on a negative edge.

Figure 3.2. ACMP Level Output used as PRS Signal for TIMER0 CC0 channel input

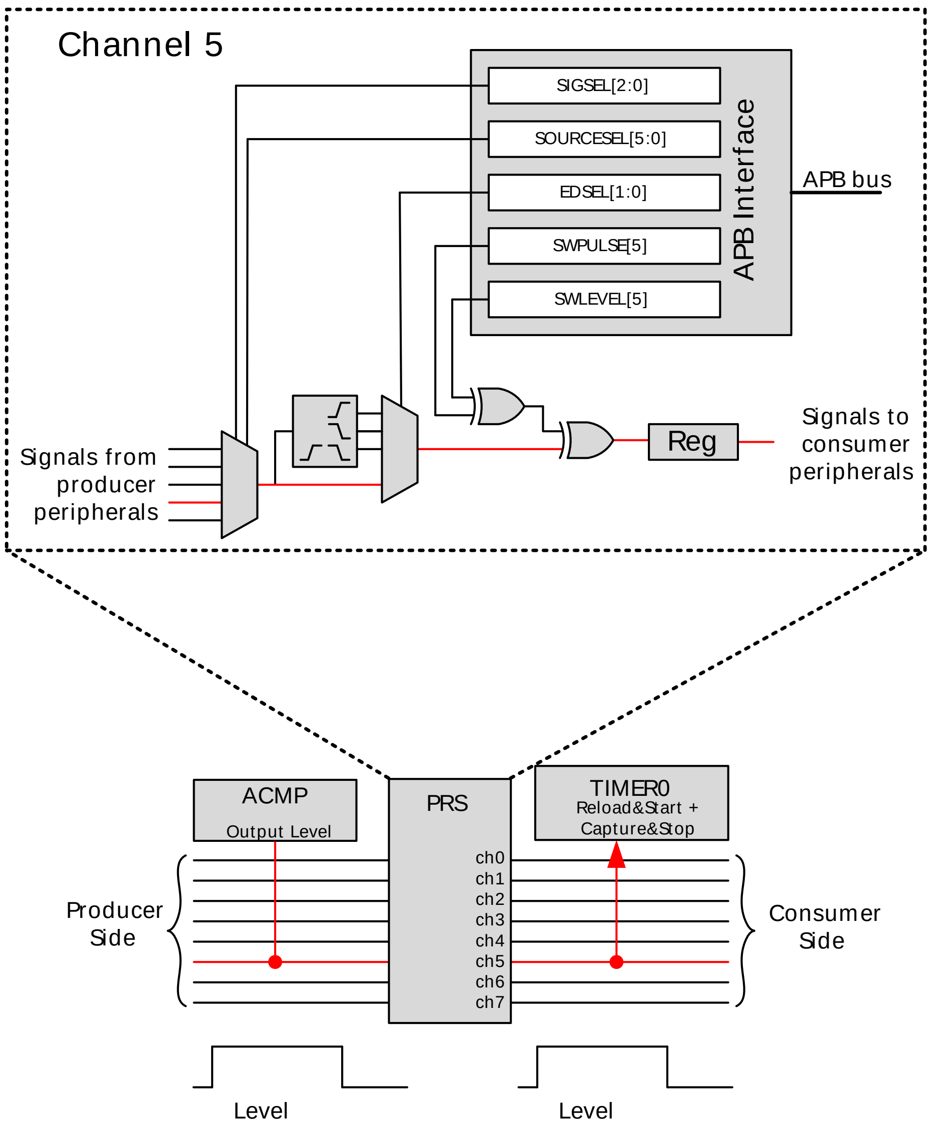


Figure 3.2 (p. 7) shows the level output from the ACMP sent through the PRS to the TIMER which measures the pulse width using the capture feature.

The software project `prs_pulse_width` implements this example and can be used on both STKs and DK. To trigger the pulse width measurement pin PC4 (P4.7 on the DK protoboard) 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 it on the LCD.

3.3 GPIO Triggered USART Transmission

This example (illustrated in Figure 3.3 (p. 8)) shows how to use an external signal coming through the GPIO to enable the USART transmitter.

Figure 3.3. USART TX enabled by GPIO signal using the PRS

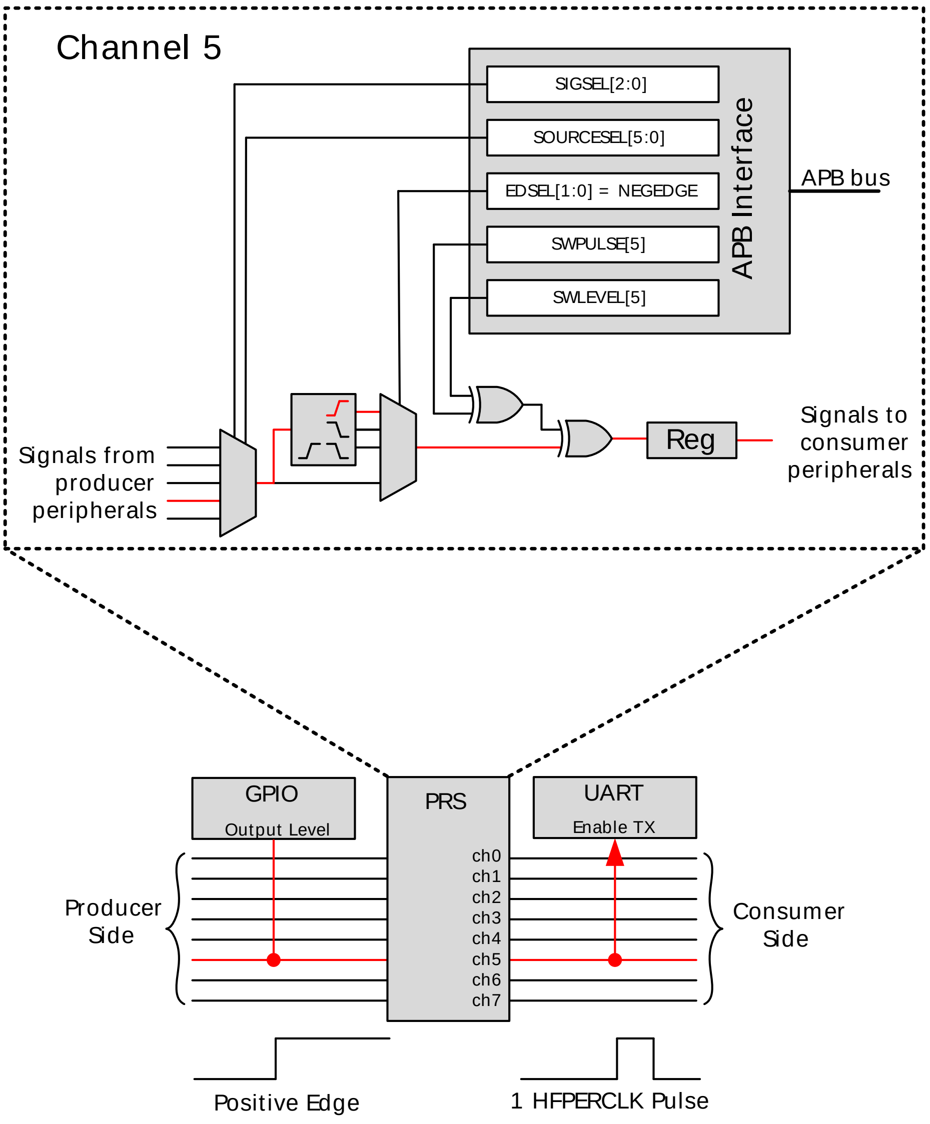


Figure 3.3 (p. 8) shows a positive edge from a GPIO pin sent on the producer side through the PRS edge detector to create a one HFPERCLK cycle pulse on the consumer side. The GPIO produces level signals which are not consumed by the UART so the edge detector must be used to generate a pulse signal on a GPIO positive edge transition. The clock pulse enables the USART TX and transmits the data that was placed in the TX buffer. For the GPIO to generate PRS signals the PRS Sense must be enabled in the GPIO_INSENSE register.

The software project `prs_gpio_uart` implements this example and is intended for the DK only. To enable the USART TX pin PD0 must be connected to VMCU to generate a positive edge. The EFM32 will then send the character 'X' through SERIAL A with a 57600 baud rate, no parity and one stop bit.

3.4 Software generated PRS Pulse triggers DAC conversion

In this example (illustrated in Figure 3.4 (p. 9)) it is shown how to generate a PRS pulse by software. The PRS pulse will trigger a DAC conversion which outputs a 0.5V signal on pin PB11 (P3.14 on the DK protoboard). 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.

Figure 3.4. Software Triggered PRS Signal.

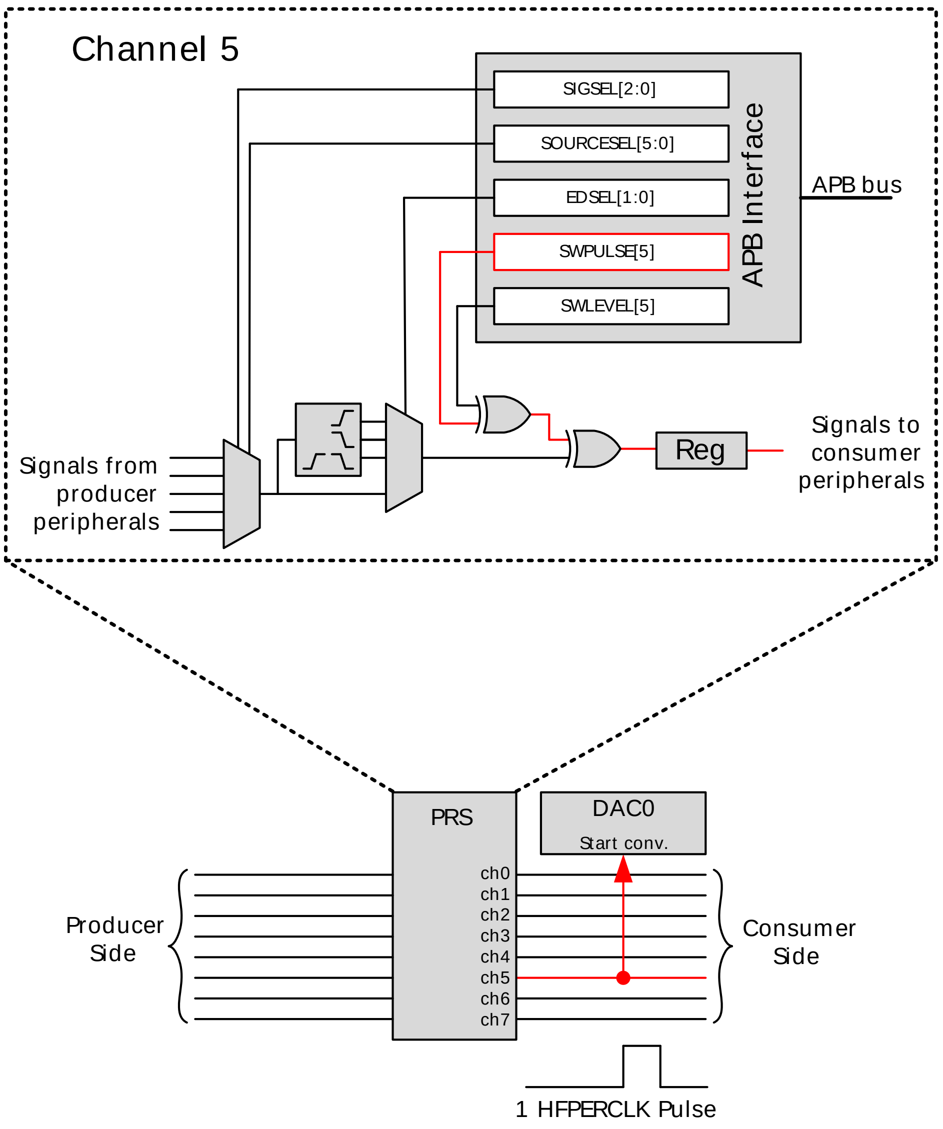


Figure 3.4 (p. 9) shows a one HFPERCLK cycle pulse triggered from software. Pulse and level signals can be generated by software by writing directly to the PRS_SWPULSE and PRS_SWLEVEL registers respectively. They can also be generated using functions from the emlib:

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

The software project `prs_soft_dac` implements this example and can be used on both STKs and DK.

3.5 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 software project `main_prs_channel_scan` exemplifies how this can be accomplished and can be used on both STKs and DK.

The function `PRS_ScanChannel(TIMER_TypeDef *timer, TIMER_PRSSEL_TypeDef prsCh, TIMER_Edge_TypeDef edgeType)` in the `main_prs_channel_scan` project can be used to monitor activity on a specific PRS channel. It sets the CC0 channel on the chosen TIMER to capture a selected signal edge. The project can be used on both STKs and DK and the parameters are as follows:

- `timer`: pointer to the TIMER peripheral register c\block
- `prsCh`: PRS channel to be monitored
- `edgeType`: Signal edge to be monitored/captured

This function will hang on a while loop waiting for activity in the PRS channel. When such activity occurs it writes PRS and the channel number on the LCD. To generate activity on this line the user must connect PC4 (P4.7 on the DK protoboard) to VMCU to generate a rising edge transition on the PRS channel using the ACMP.

Another option is to have a capture interrupt instead of polling. This way the program will not hang and the processor will be available to execute other tasks. When a capture is triggered the user knows that there was activity on the selected PRS channel.

4 Revision History

4.1 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

4.2 Revision 1.06

2013-10-14

New cover layout

4.3 Revision 1.05

2013-05-08

Added software projects for ARM-GCC and Atollic TrueStudio.

4.4 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.

4.5 Revision 1.03

2012-04-20

Adapted software projects to new peripheral library naming and CMSIS_V3.

4.6 Revision 1.02

2011-10-21

Updated IDE project paths with new kits directory.

4.7 Revision 1.01

2011-05-18

Updated projects to align with new bsp version.

4.8 Revision 1.00

2010-12-13

Initial revision.

A Disclaimer and Trademarks

A.1 Disclaimer

A.2 Trademark Information

B Contact Information

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

Please visit the Silicon Labs Technical Support web page:
http://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.

Table of Contents

List of Figures

List of Tables

silabs.com

ZERO
ARM Cortex-M0+

TINY
ARM Cortex-M3

GECKO
ARM Cortex-M3

LEOPARD
ARM Cortex-M3

GIANT
ARM Cortex-M3

WONDER
ARM Cortex-M4