CC254x Basic Software Examples Overview

These software examples are designed with primarily the CC2541, CC2543, CC2544 and CC2545 System on Chips (SoC) in mind. The examples use the Evaluation Modules (EM) and SmartRF05 Evaluation Board (EB) rev. 1.8.1 as well as the CC2544Dongle for the CC2544 device. The IDE used is the IAR Embedded Workbench for 8051, version 8.11 This version might need some additional patches downloadable from the IAR web site. For further information on IAR project setup or the SmartRF05EB see the SmartRF05EB User's Guide. A SmartRF05EB pinout is included in this document. This overview explains the setup for CC2543EM on the SmartRF05EB, the same behaviour is implemented for the other devices with very small differences (read the comments in the source code for more details). Also read the "How_To_Change_Device_in_IAR_Workspace.pdf" document to learn how to change between the devices in the IAR work space.

Module/Project	Example	Description	Notes
ADC	adc_cpu_poll_single	This example performs single-ended, single channel, CPU controlled ADC conversion. The potmeter on SRF05EB can be used to source the analog signal by connecting PIN 17 on the P18 Debug Connector, to PIN 12 on the P20 Debug Connector.	ADC configuration: - Single-ended and single-channel on PIN0_0 - Reference Voltage is VDD on the AVDD pin - 12 bit resolution (512 dec rate) - The ADC is not applicable for the CC2544
	adc_extra_isr_single	The example illustrates how the ADC interrupt works, based on extra conversion trigger. The potmeter on SRF05EB can be used to source the analog signal by connecting PIN 17 on the P18 Debug Connector, to PIN 12 on the P20 Debug Connector.	ADC configuration: - Single-ended and single-channel on PIN0_0 - Reference Voltage is VDD on the AVDD pin - 10 bit resolution (256 dec rate) - The ADC is not applicable for the CC2544
	adc_cont_dma_seq	This example shows how the DMA can be used to store the results from an ADC conversion. It performs single-ended ADC sequence in continous mode with DMA transfer on end of ADC conversion event.	ADC configuration: - Continous mode - Single-ended sequence conversion (P0_0 - P0_1) - Reference Voltage is VDD on the AVDD pin - 9 bit resolution (128 dec rate)
CLK	clk_hs_xosc	This example changes the system clock source to High Speed Crystal Oscillator (HS XOSC) and sets clock speed to highest (32 MHz).	Disables the 32 kHz RCOSC calibration, prior to the clock change.
	clk_hs_rcosc	This example calibrates the High Speed RC oscillator (HS RCOSC) and the 32 kHz RC oscillator (LS RCOSC), and sets the system clock source to HS RCOSC with clock speed = 2 MHz.	The debugger cannot be used with a divided system clock.
	clk_ls_xosc	This examples sets the 32 kHz clock source to 32.768 kHz Crystal Oscillator (LS XOSC).	The LS XOSC is only available on the CC2541 and the CC2545 devices.
DMA	dma_man_trigger	This example shows how to trigger the DMA manually and make the DMA move data from one area of RAM to another.	-
	dma_prev_trigger	This example shows both how to trigger a DMA channel manually and how to trigger a DMA channel by the completion of the previous channel. Both channels copy data in RAM. Channel 0 makes the first copy and channel 1 makes a copy of the first copy.	-
FLASH	flash_dma	This example illustrates how to write data to flash memory using DMA. The DMA transfer method is the preferred way to write to the flash memory, since when using the DMA to write to flash, the code can be executed from within flash memory.	-
I^2C	i2c_master_send	This example uses a Master to send data to a Slave using I^2C. Acknowledge is enabled and the most basic bus states are presented. The send and receival routines are based on interrupts. When the transmission is finished LED1 is set.	To execute the example use anothor device running i2c_slave_receive as the receiver.
	i2c_slave_receive	This example uses a Slave to receive data from a Master using I^2C. Acknowledge is enabled and the most basic bus states are presented. The send and receival routines are based on interrupts. When the transmission is finished LED1 is set.	To execute the example use anothor device running i2c_master_send as the transmitter.
	i2c_master_receive	This example uses a Master to receive data from a Slave using I^2C. Acknowledge is enabled and the most basic bus states are presented. The send and receival routines are based on interrupts. When the transmission is finished LED1 is set.	To execute the example use anothor device running i2c_slave_send as the transmitter.
	i2c_slave_send	This example uses a Slave to send data to a Master using I^2C. Acknowledge is enabled and the most basic bus states are presented. The send and receival routines are based on interrupts. When the transmission is finished LED1 is set.	To execute the example use anothor device running i2c_master_receive as the receiver.
	io_out	This examples sets up P0_0 to generate interrupts when output is changed.	Button S1 on SmartRF05EB is used to toggle P0_0. When a rising edge is detected on the pin, the ISR for the pin will be executed. This will toggle LED1.

Module/Project	Example	Description	Notes
IO	io_multi_in_out	This examples uses multiple pins to communicate with each other. P2_0 is configured as output and P1_4 is configured as input.	Button S1 on SmartRF05EB is used to toggle P2_0. P1_4 will generate interrupt on incoming event. So, by connecting P2_0 and P1_4 (with a wire), an interrupt will be generated when the button is pushed. This will toggle LED1 .
	io_in	This example sets up one pin on port 1 to generate interrupts on incoming events. When a rising edge is detected on the pin, the ISR for the pin will be executed. This will toggle LED1 on SRF05EB. The pin can be chosen using the PIN macro.	The pin that is set up, can be toggled with GND, in example using a wire from GND to the pin.
PowerMode (PM)	idle	This example shows how to correctly enter idle mode and wake up using an External Interrupt. The example uses Button 1 (S1) on SmartRF05EB to generate the External Interrupt. For test purpose, LED 1 is used to show when the SoC enters and exits idle mode. When LED 1 is set, the SoC will stay in Active Mode for some time before going into idle again, when the LED is cleared.	Settings: -Interrupt: External Interrupt -System clock oscillator: HS XOSC oscillator -Clock speed: 32 MHz
	pm1	This example shows how to correctly enter Power Mode 1 and wake up using an External Interrupt. The example uses Button 1 (S1) on SmartRF05EB to generate the External Interrupt. For test purpose, LED 1 is used to show when the SoC enters and exits PM1. When the LED 1 is set, the SoC will stay in Active Mode for some time before entering PM1 again, when the LED is cleared.	Settings: -Interrupt: External Interrupt -System clock oscillator: HS XOSC oscillator -Clock speed: 32 MHz In addition an assembly function is created to enter the power mode correctly.
	pm2	See Sleep Timer example.	-
	pm3	This software example shows how to properly enter Power Mode 3 and then exit upon Port 0 Interrupt. The SRF05EB LED1 is cleared before entering Power Mode 3, and then set by the Port 0 ISR. Hence, the SRF05EB LED1 is ON in Active Mode and OFF in PM3.	Uses Port 0 Interrupt to exit PM3, The default system clock HS RCOSC at 16 MHz is used. In addition an assembly function is created to enter the power mode correctly.
Random Number Generator	rnd_adc	This example generates a random number based on an ADC conversion. The ADC is set up for single, low resolution conversion on the temperature sensor. The result is used to seed the random number generator, which then clocks the LSFR once and hence generates a random number.	The random number is stored as the variable rndNumber which can be read in debug mode. Note that the seed may be made more random if that is required.
	rnd_rf	This example sets up a true-random number generator using radio noise measurements. The radio is set up in RX mode without sync, and the Radio ADC seeds the Random Number Generator.	An array, "rndArray" stores the random numbers and can be viewed using the debugger.
Sleep Timer	sleep_timer	This example shows how to setup the Sleep Timer for using it as the source for waking up from a Power Mode. In the example the system goes into Power Mode 2 after setting up the Sleep Timer and Sleep Timer interrupt. The Sleep Timer is set up to generate an interrupt every second. When awaken, the ISR toggles the LED1.	The clock source for the Sleep Timer is the 32.753 kHz LS RCOSC
SPI	spi_master_send	This example uses a master to send data to a slave using SPI. The bytes sent are simply numbers 0x00 upto BUFFER_SIZE. When bytes upto BUFFER_SIZE are sent, the example will end and LED1 will be set on SmartRF05EB.	To execute this example use the IAR project spi_slave_receive as the slave. The slave's code must be executed before executing the master's code, since the slave is clocked by the Master. Note that if BUFFER_SIZE is larger than 0xFF, than the element 0xFF in the array will have the value 0x00 and so on.
	spi_slave_receive	This example uses a slave to receive data from a master using SPI. When bytes upto BUFFER_SIZE have been received, the example will end and LED3 will be set on SmartRF05EB.	To execute this example, use the IAR project spi_master_send , as the master. The slave's code must be executed before executing the master's code, since the slave is clocked by the master. You can debug the rxBufferSlave array to verify that the bytes received are the correct ones and are in correct order.
	spi_master_receive	This example uses a master to receive data from a slave using SPI. When bytes upto BUFFER_SIZE are received, the example will end and LED1 will be set on SmartRF05EB.	To execute this example, use the IAR project spi_slave_send, as the slave. The slave's code must be executed before executing the master's code, since the slave is clocked by the master. You can debug the rxBufferMaster array to verify that the bytes received are the correct ones and are in correct order.
	spi_slave_send	This example uses a slave to send data to a master using SPI. The bytes sent are simply numbers 0x00 upto BUFFER_SIZE. When bytes upto BUFFER_SIZE are sent, the example will end and LED3 on SmartRF05EB will be set.	To execute this example, use the IAR project spi_master_receive, as the master. The slave's code must be executed before executing the master's code, since the slave is clocked by the master. Note that if BUFFER_SIZE is larger than 0xFF, than the element 0xFF in the array will have the value 0x00 and so on.

Module/Project	Example	Description	Notes
	spi_slave_receive_isr	This example uses a slave to receive data from a master using SPI and interrupts. When the buffer is full, LED3 on SmartRF05EB is set.	To execute this example, you can use the IAR project spi_master_send, as the master. Note that the slave's code must be executed before executing the master's code, since the slave is clocked by the master. If more data is received after this, the new data will overwrite the previous data in buffer. You can debug the rxBufferSlave array to verify that the bytes received are correct.
Timer1	t1_updown	This example uses Timer 1 in up/down mode while toggling P0_4 and generating [Timer 1 Channel 2] interrupt when counter reaches compare value, and [Timer 1 Channel 0] interrupt when counter wraps around zero. LED1 is toggled when channel 2 interrupt occurs, while LED3 is toggled when channel 0 interrupt occurs. The variables timer1Ch2CmpResult and timer1Ch0CmpResult can be used for debugging.	Settings: - Default clock source (HS RCOSC at 16 MHz) - Up/Down mode - 125 kHz Timer 1 tickspeed - Output compare mode with toggling on compare - Ch2 interrupt when counter reaches compare value. - Ch0 interrupt when counter turns around on zero.
	t1_mod	This example uses Timer 1 to function in Modulo mode. Channel 0 is used to toggle LED3 at 2 Hz. To acheive this we need to prioritize Timer 1 and set I/O location to alternative 2. Channel 0 needs to be in compare mode, and toggle output on compare. The Timer tickspeed needs to be set at 125 kHz, with a terminal count value at 62500.	Settings: - Default clock source (HS RCOSC at 16 MHz) - Modulo mode - 125 kHz tickspeed - Output compare mode with toggling on compare - Terminal count value = 62500
	t1_free	This example uses Timer 1 to function in Free Running mode. Channel 0 is used to toggle P0_2 when counter reaches compare value on channel 0. An interrupt is generated when the Timer 1 counter reaches compare value. LED1 is toggled when channel 0 interrupt occurs. The LED should blink about 3 times per second because of the frequency used. The variable timer1Ch0CmpResult can be used for debugging.	Settings: - Default clock source (HS RCOSC at 16 MHz) - Free-running mode - 500 kHz tickspeed - Channel 0 - Output compare mode with toggling on compare - Interrupt generated on compare
	t1_cap	This example uses Timer 1 to function in Free Running mode. Channel 0 is used to capture events from P0_2 on falling edge. A channel 0 interrupt is generated when an event is captured. In the ISR, LED1 is toggled.	Settings: - Default clock source (HS RCOSC at 16 MHz) - Free-running mode - 500 kHz tickspeed - Channel 0 - Input capture on P0_2 - Interrupt generated on capture
Timer3/4	t3_modulo	This example runs Timer 3 in modulo mode. At each compare match the terminal count value is altered, and LED1 is toggled.	Timer 3 configuration: - Prescaler divider value: 64. - Modulo mode. - Compare mode. - Channel 0 interrupt enabled.
	t3_free	This example runs Timer 3 in Free running mode. LED1 is toggled on each overflow.	Timer 3 configuration: - Prescaler divider value: 128. - Overflow interrupt enabled. - Free running mode.
	t3_down	This example runs Timer 3 in down mode. Initially, LED1 is turned on. When the Timer has counted down to 0x00, an interrupt occurs, it turns LED1 off and LED3 on.	Timer 3 configuration: - Prescaler divider value: 128. - Overflow interrupt enabled. - Down mode.
	t4_pwm	This example generates a sine wave signal, by using Timer 4 to generate a PWM signal with varying duty cycle. The sine wave is set to 4 Hz and manipulates LED1.	The frequency, f, of the sine wave is: f = f_tts / (t4_div * 2 * t4_tcv * N) where f_tts is the global timer tick speed, t4_div is the Timer 4 prescaler divider value, t4_tcv is the Timer 4 terminal count value and N is the number of samples per sine wave.
UART	uart_poll	This example sends/receives data on UART0, using polling method. Pressing S1 makes the transmitter send its allocated uartTxBuffer[] to the receiver. The transferred data will arrive at uartRxBuffer[] and can be verified using the debugger.	To execute this example, compile one SmartRF05EB unit as transmitter, by activating the UART_TST_MODE_TX definition, then compile another unit as receiver, by activating the UART_TST_MODE_RX definition.
	uart_isr	This example sends/receives data on UART0, using interrupt method. Pressing S1 makes the transmitter send its allocated uartTxBuffer[] to the receiver. The transferred data will arrive at uartRxBuffer[] and can be verified using the debugger.	To execute this example, compile one SmartRF05EB unit as transmitter, by activating the UART_TST_MODE_TX definition, then compile another unit as receiver, by activating the UART_TST_MODE_RX definition.
	uart_dma	This example sends/receives data on UART0, using DMA method. Pressing S1 makes the transmitter send its allocated uartTxBuffer[] to the receiver. The transferred data will arrive at uartRxBuffer[] and can be verified using the debugger.	To execute this example, compile one SmartRF05EB unit as transmitter, by activating the UART_TST_MODE_TX definition, then compile another unit as receiver, by activating the UART_TST_MODE_RX definition.
Watchdog Timer (WDT)	wdt_wd	This example uses the WDT in Watchdog mode.	The Watchdog causes a reset approx. every second. The LEDs indicate what caused the reset (read from [SLEEPSTA.RST]).
	wdt_timer	This example uses the WDT in Timer mode to toggle a LED on Timer overflow interrupt.	The low power LS RCOSC is used (default); the accuracy of timeout interval will vary with the choice of low speed OSC, see the datasheet for details.

Pinout SmartRF05EB rev. 1.8.1

Debug Connector	Signal	CC2541EM R1.1	CC2543EM R1.1	CC2545EM R1.1
P18_01				
P18_03	USB1	I2C SDA		P2.6/I2C SDA
P18_05	USB2	I2C SCL		P2.5/I2C SCL
P18_07	EM_BUTTON1/EM_LED4_SOC	P0.1	P0.1	P0.1
P18_09	EM_UART_RX	P0.2/UART RX	P0.2/UART RX	P0.2/UART RX
P18_11	EM_UART_TX	P0.3/UART TX	P0.3/UART TX	P0.3/UART TX
P18_13	EM_UART_CTS	P0.4/UART CTS	P0.4/UART CTS	P0.4/UART CTS
P18_15	EM_UART_RTS	P0.5/UART RTS	P0.5/UART RTS	P0.5/UART RTS
P18_17	EM_POT_R	P0.7		P0.7
P18_19	EM_DBG_DIR	-	-	-
P18_02				
P18_04	EM_FLASH_CS	P1.3		**P3.7**
P18_06	EM_LED2_SOC	P1.1		P1.1
P18_08	EM_DBG_DD	P2.1/DD	P2.1/DD	P1.3/DD
P18_10	EM_DBG_DC	P2.2/DC	P2.2/DC	P1.4/DC
P18_12	EM_MISO	P1.7/SPI MISO	P1.3/SPI MISO	P2.3/SPI MISO
P18_14	EM_CS/EM_LED3_SOC	P1.4/SPI CSn	P1.1/SPI CSn	P2.1/SPI CSn
P18_16	EM_SCLK	P1.5/SPI SCLK	P1.2/SPI SCLK	P2.2/SPI SCLK
P18_18	EM_MOSI	P1.6/SPI MOSI	P1.4/SPI MOSI	P2.7/SPI MOSI
P18_20	GND	GND	GND	GND
P20_01				
P20_03	VCC_EM	VDD	VDD	VDD
P20_05	EM_PWR_SNOOZE			
P20_07	EM_JOYSTICK_RT			P3.2
P20_09	EM_JOYSTICK_DN			P3.0
P20_11	EM_JOYSTICK_UP			P3.1
P20_13	EM_JOYSTICK_LT			P3.3
P20_15	EM_JOYSTICK_PUSH			**P3.6**
P20_17	EM_JOY_LEVEL	P0.6	P0.6/I2C SCL	P0.6
P20_19	EM_JOY_MOVE	P2.0	P2.0	P2.0
P20_02				
P20_04	EM_LED1	P1.0	P1.0	P1.0
P20_06	EM_LED2_MSP			P3.4
P20_08	EM_LED3_MSP			P3.5
P20_10	EM_LED4_MSP			
P20_12	EM_LCD_MODE	P0.0	P0.0	P0.0
P20_14	EM_RESET	RESET	RESET	RESET
P20_16	EM_BUTTON2			P2.4
P20_18	EM_LCD_CS	P1.2	P0.7/I2C SDA	P1.2
P20_20	GND	GND	GND	GND

Pinout CC2544Dongle rev. 1.0

Debug Connector	Signal	CC2544Dongle
P2_1		P1_3
P2_2		P1_2
P2_3		P1_1
P2_4		P1_0
P3_1	Button 2 (S2)	P0_3
P3_2	LED2 (D2 - GREEN)	P0_2
P3_3	LED1 (D1 - RED)	P0_1
P3_4	Button 1 (S1)	P0_0
DEBUG Header Pin:		
1	GND	GND
3	DC	P1.2
5		
7	RESET_N	RESET_N
9		
2	VDD	VDD
4	DD	P1.3
6		
8		
10		

References:

[1] CC254x System-on-Chip Solution for 2.4-GHz Applications, User's Guide (SWRU283A)
[2] SmartRF05 Evaluation Board, User's Guide (SWRU210A)
[3] CC2541 System-on-Chip for 2.4-GHz RF Applications, Datasheet (SWRS110B)
[4] CC2543 System-on-Chip for 2.4-GHz RF Applications, Datasheet (SWRS107B)
[5] CC2544 System-on-Chip for 2.4-GHz RF Applications, Datasheet (SWRS103D)
[6] CC2545 System-on-Chip for 2.4-GHz RF Applications, Datasheet (SWRS106A)
[7] CC2541EM Reference Design (SWRR096)
[8] CC2543EM Reference Design (SWRR089)
[9] CC2544 USB Dongle Reference Design (SWRR099)
[10] CC2545EM Reference Design (SWRR104)