Spansion Laboratory Training

V1.4 Jan 29, 2015

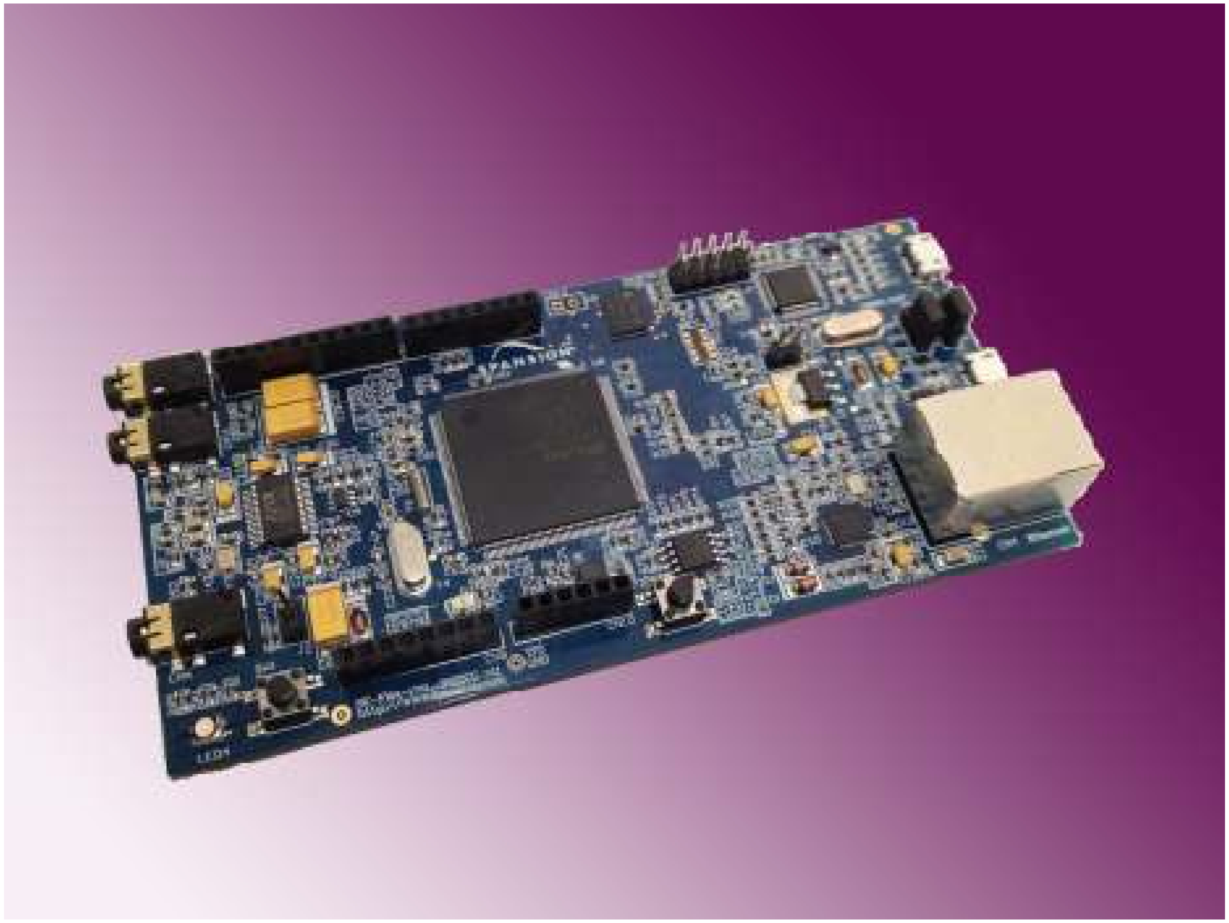
FM4

V1.1 24.07.2013

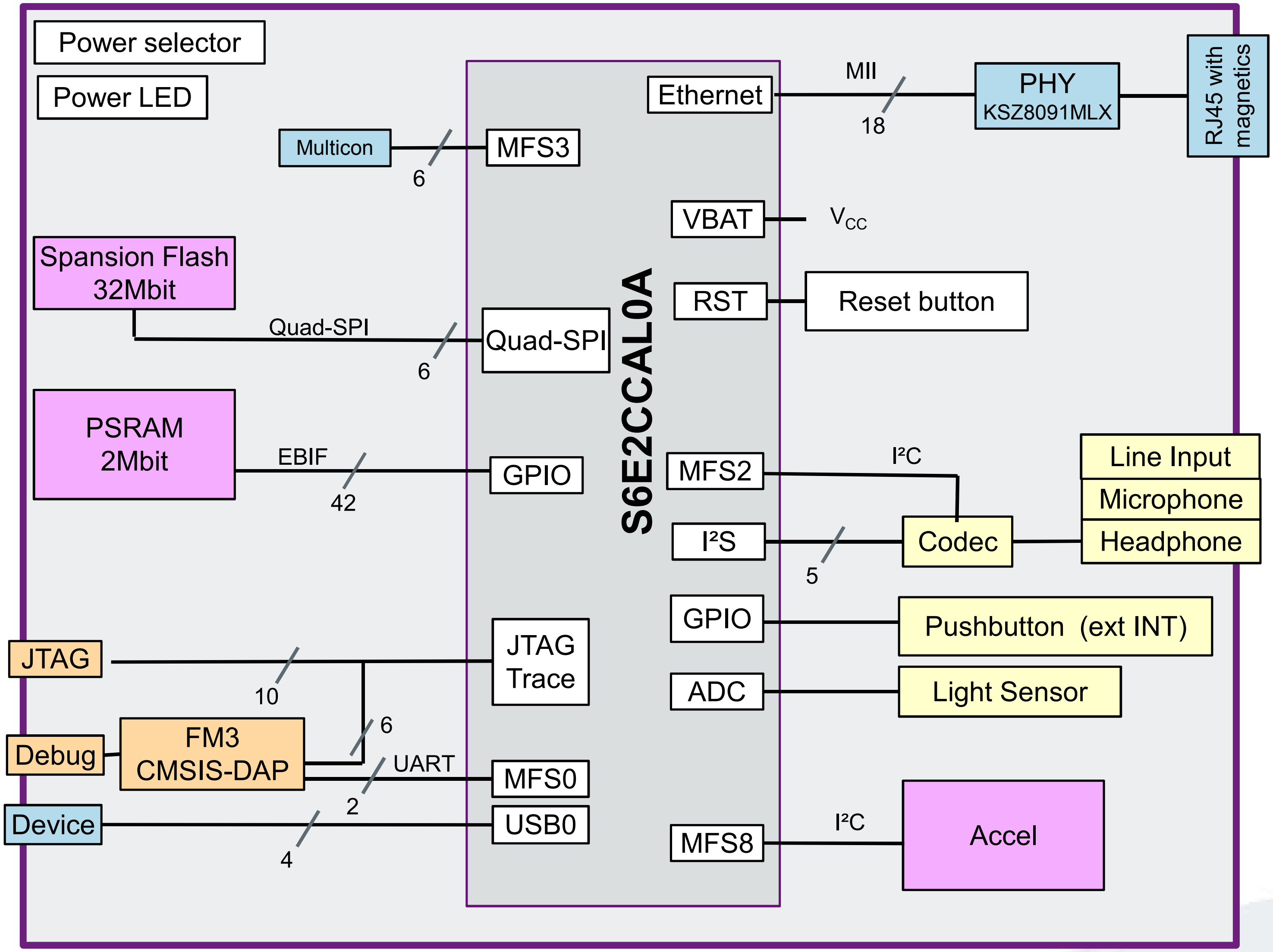
SK-FM4-176L-S6SE2CC Starter Kit

Introduction and Features
Arduino Interface
Quick Start Guide - Board Test Software
Ordering Information

SK-FM4-176L-S6SE2CC Board Photo



SK-FM4-176L-S6SE2CC Block Diagram



  CONFIDENTIAL

SK-FM4-176L-S6SE2CC Features List

Cat	Feature	Number	Why	How
Comms I/F	Ethernet	1		Micrel PHY KSZ8091MLX
	Arduino interface	1	Ability to add new functionality	
	Multicon	1	Additional standardized serial connection	2*5 pin header, same as on other SKs
	USB Device	1	Micro AB OTG connector but DEVICE only	
Debug I/F	JTAG	1	10-pin interface	
		1		
	On-board JTAG	1	J-Link-OB or CMSIS-DAP (firmware option)	**MB9BF312K**
Memory	Ext. Quad SPI Flash -32MB	1	3V3	**S25FL032P**
	PSRAM 2MB	1	3V3	SV6P1615UFC
User I/F	Accelerometer	1		KXCJK-1013
	User button	1	H/W with IRQ, needed for user interface	Small button
	I²S	1	Audio or speech	WM8731SEDS
	Light sensor	1	User ADC input	PT11-21C/L41/TR8
	RGB LED	1	User feedback	CLV1A-FKB-CJ1M1F1BB7R4S3
Misc				
	MCUVCC	3V3		
	Board voltage source selector	1	CMSIS, JTAG, MCU USB	selectable via jumper
	Reset button	1		

SK-FM4-176L-S6SE2CC Starter Kit Introduction

SK-FM4-176L-S6SE2CC Key Components

Name	Part	Description
MCU	Orion (FM4 series)	LQFP-176 (0.5mm)
Flash	Spansion, S25FL032P	32Mbit, 104 MHz, qSPI
PSRAM	SV6P1615UFC	1M*16bit, 48FBGA
CMSIS-DAP	MB9AF312K	LQFP-48
Codec	WM8731CLSEFL	Microphone, Headphone, Line-in
Acceleration Sensor	ROHM	3D Acceleration Sensor, analog output
Ethernet PHY	KSZ8091MNXCA	MII, 10-100M Ethernet PHY

SK-FM4-176L-S6SE2CC Starter Kit Introduction

SK-FM4-176L-S6SE2CC Connector and Jumper Locations

LED assignments	
LED1	CMSIS-DAP status – Yellow-Green
LED2	CMSIS-DAP - red
LED3	Board power – Yellow-Green
LED4	Tricolor – Under user control

J1 – CMSIS DAP programming	
P1 – P2 =	MD0 tied to 3V3 – DUT in BIROM mode
Open =	User mode

J2 - DUT mode select	
P1 – P2 =	MD0 tied to 3V3 – DUT in BIROM mode
Open =	User mode

J3 – BIROM mode select	
P1 – P2 = {Not supported}	Port pin P60 tied to ground (Sets UART BIROM)
P2 – P3 = ** Leave in this position for labs	Port pin P60 tied to USB Vbus

J4 – Power select	
P1 – P2 =	DAP feeds 3V3 Regulator
P2 – P3 =	fUSB5V needs 3V3 Regulator

SK-FM4-176L-S6SE2CC Starter Kit Introduction

SK-FM4-176L-S6SE2CC Connectors to Arduino and Multicon Interfaces

Pin Assignment of S6E2CC Starterkit connectors				
Pin No.	**Pin Name**	**Arduino I/F**	**Arduino Connector designation**	**Multicon**
34	P38/ADTG_2/DTTI0X_0/S_WP_0		CN7 - pin 1	
46	P40/SIN3_1/RTO10_0/TIOA0_0/AIN0_0/INT23_0/MCSX7_0	DIG10-6/SPI_SI/MFT_PWM	CN7 - pin 4	CN12 - pin 7
47	P41/SOT3_1/RTO11_0/TIOA1_0/BIN0_0/MCSX6_0	DIG10-7/SPI_SO/MFT_PWM	CN7 - pin 5	CN12 - pin 2
48	P42/SCK3_1/RTO12_0/TIOA2_0/ZIN0_0/MCSX5_0	DIG10-8/SPI_CLK/MFT_PWM	CN7 - pin 6	CN12 - pin 1
49	P43/SIN15_0/RTO13_0/TIOA3_0/INT04_0/MCSX4_0	DIG8-4/MFT-PWM	CN9 - pin 4	CN12 - pin 6
50	P44/SOT15_0/RTO14_0/TIOA4_0/MCSX3_0	DIG8-6/MFT-PWM	CN9 - pin 6	CN12 - pin 8
51	P45/SCK15_0/RTO15_0/TIOA5_0/MCSX2_0	DIG8-7/MFT-PWM	CN9 - pin 7	
57	INITX	Power - 4	CN-8 pin 3	
63	PF0/SCS63_0/RX2_1/FRCK1_1/TIOA15_1/INT22_1	DIG10-3/SPI_CS/PWM	CN7 - pin 3	CN12 - pin 5
71	PF3/RTO11_1/TIOB6_1/INT05_1/MCASX_0	DIG8-8/IO/BT	CN9 - pin 8	
72	PF4/RTO12_1/TIOA7_1/INT06_1/MSDWEX_0	DIG10-2/PWM	CN7 - pin 2	
73	PF5/RTO13_1/TIOB7_1/INT07_1/MCSX8_0	DIG8-3/IO/BT	CN9 - pin 5	
75	PF7/RTO15_1/TIOB14_1/INT21_1/MSDCLK_0		CN9 - pin 3	
94	P10/AN00/SIN10_0/TIOA0_2/AIN0_2/INT08_0	ANA-6	CN10 - pin 6	
95	P11/AN01/SOT10_0/TIOB0_2/BIN0_2	ANA-5	CN10 - pin 5	
96	P12/AN02/SCK10_0/TIOA1_2/ZIN0_2	ANA-4	CN10 - pin 4	
97	P13/AN03/SIN6_1/RX1_1/INT25_1	ANA-3	CN10 - pin 3	
98	P14/AN04/SOT6_1/TX1_1	ANA-2/SDA1	CN10 - pin 2	
100	P16/AN06/SOT11_0/TIOA2_2/BIN1_2	DIG10-9/SDA2	CN7 - pin 9	
101	P17/AN07/SCK11_0/TIOB2_2/ZIN1_2		CN7 - pin 10	
102	PB0/AN16/SCK6_1/TIOA9_1	ANA-1/SCK1	CN10 - pin 1	
109	P1B/AN11/SIN12_0/TIOB4_2/INT11_0/TRACED1***	DIG8-2/UART_RX	CN9 - pin 1	
110	P1C/AN12/SOT12_0/TIOA5_2/TRACED2 ***	DIG8-1/UART_TX	CN9 - pin 2	

SK-FM4-176L-S6SE2CC Starter Kit Arduino Interface

SK-FM4-176L-S6SE2CC Arduino Interface

Starter Kit can use same Arduino Shield boards as those designed for the well-known standard UNO3

This same pinout is used for many other Arduino boards including the LEONARDO, Seeeduino, Bugduino, and countless others.

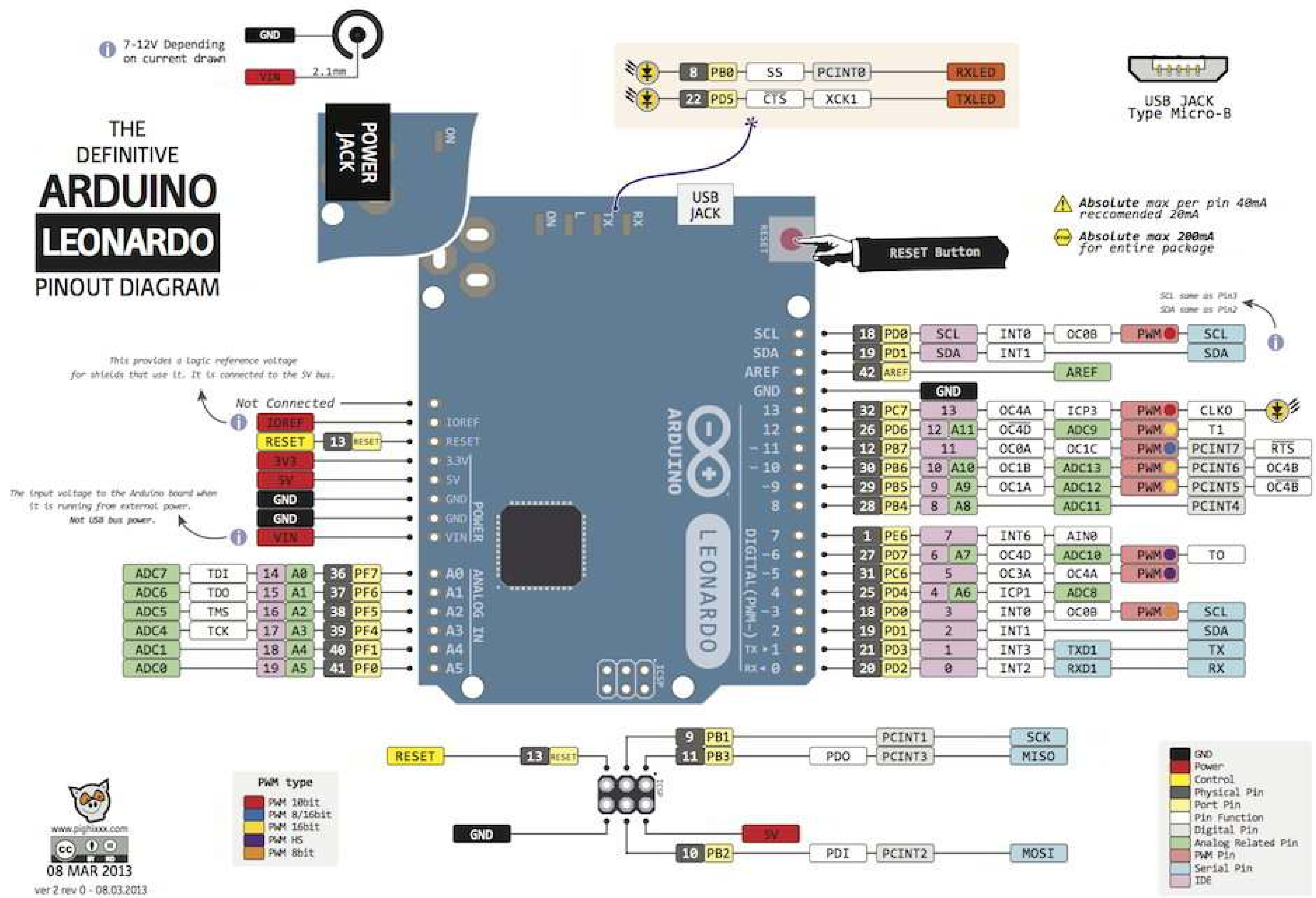
Not compatible with larger (and less popular) MEGA Shields

UNO3 Arduino

SK-FM4-176L-S6SE2CC Starter Kit Arduino Interface

SK-FM4-176L-S6SE2CC Arduino LEONARDO interface.

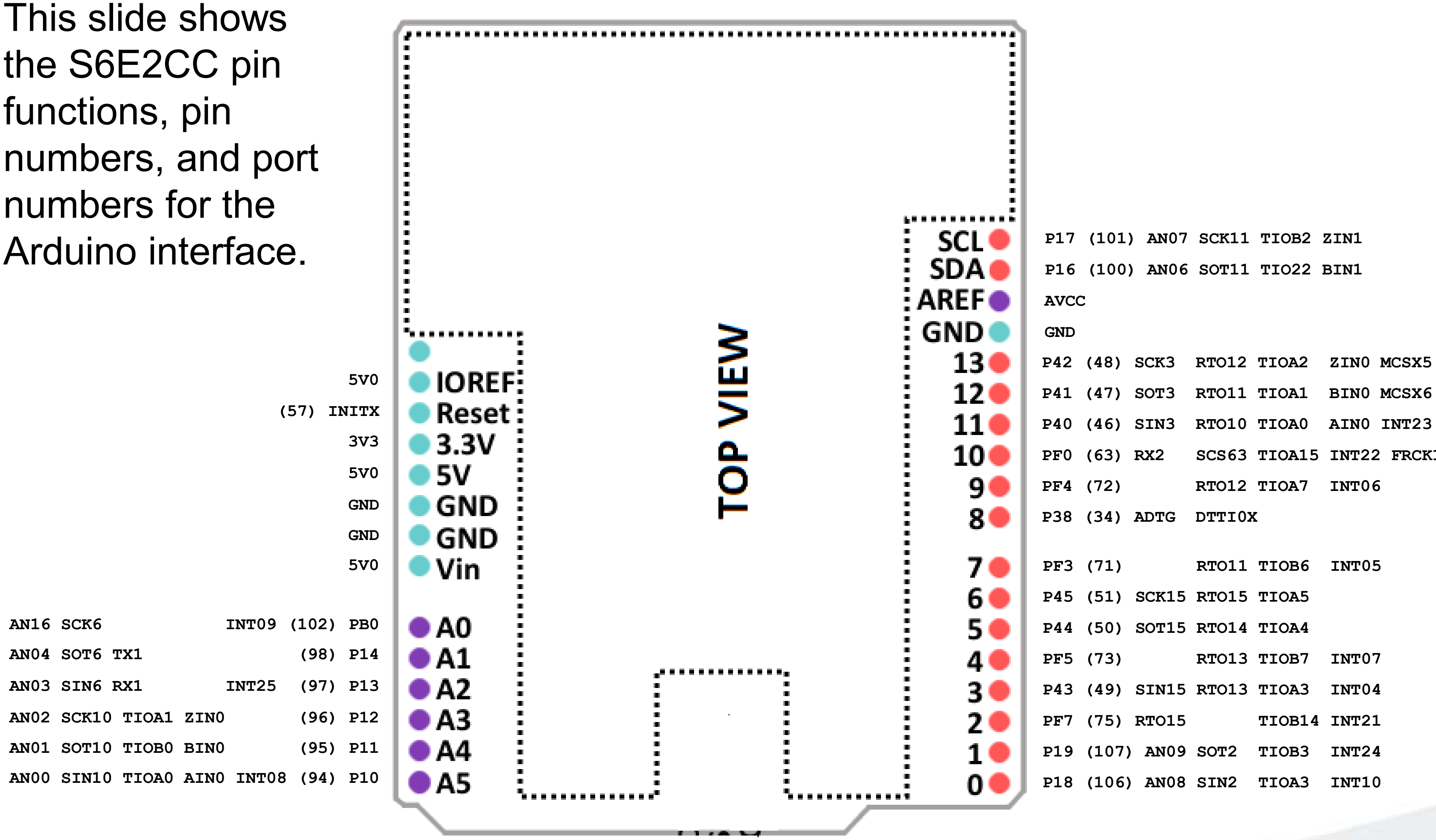
This slide shows the Arduino pinout and the multiple functions on each pin. This makes this standard much more useful that it would first appear.



SPANSION

SK-FM4-176L-S6SE2CC Reference Shield Pinout

This slide shows the S6E2CC pin functions, pin numbers, and port numbers for the Arduino interface.



SK-FM4-176L-S6SE2CC Quick Start Guide - Board Test Software

SK-FM4-176L-S6SE2CC Quick Start

Please refer to the externally provided Quick Start Guide for the procedures for installing necessary drivers and program software.

This will guide through setup to running the preprogrammed factory test software.

If it is desired to return the kit to its original factory condition the file SK-176-s6e2ccTestCode-V10.srec is provided in the .\tools subdirectory on the CD or master ZIP file.

Instructions on programming this file is contained in the Flash Programming Tools presentation.

SPANSION

Spansion®
FM4 Starter Kit

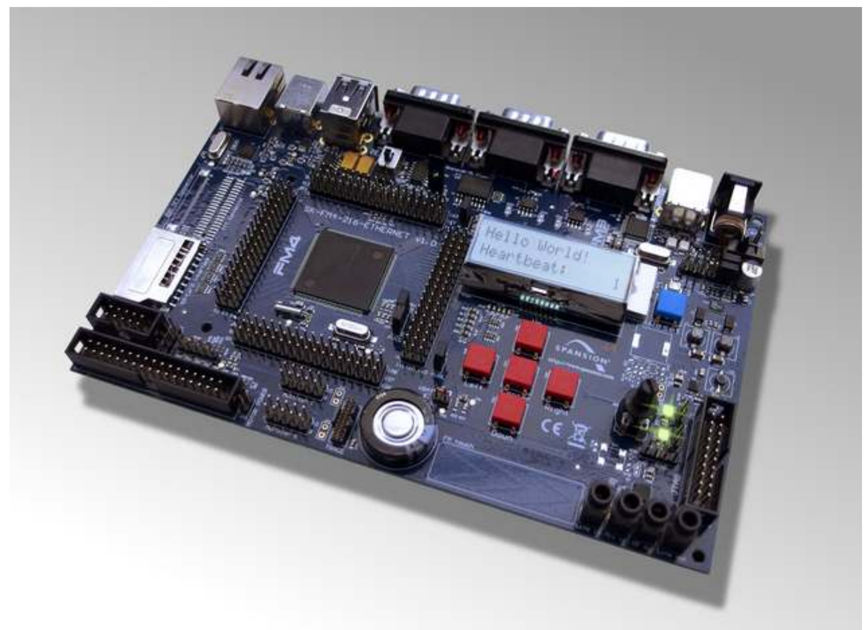
SK-FM4-176L-S6E2CC
SK-FM4-176L-S6E2CC-ETH
SK-FM4-176L-S6E2CC-VOI

QUICK START GUIDE

SK-FM4-176L-S6SE2CC Ordering Information

Ordering – Kits will soon be stocked at Digi-key and Mouser

SK-FM4-176L-S6SE2CC	-Kit with no Ethernet nor Voice MCU
SK-FM4-176L-S6SE2CC-ETH	-Kit with Ethernet, but no Voice MCU
SK-FM4-176L-S6SE2CC-VOI	-Kit with Ethernet and Voice MCU
SK-FM4-216-ETHERNET (EVB)	-Full EVB. Please see Web for details



A2

SK-FM4-176L-S6SE2CC Starter Kit

SK-FM4-176L-S6SE2CC Quick Start Closing Information

- Introduction and Features
- Explained the Arduino Interface and why it is included
- Worked through the Quick Start Guide installing needed files
- Completed a board test to make sure the hardware is good.
- Provided Ordering Information

Slide 14

A2 Author, 12/15/2014

SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Dual Flash / Programming via RAM

SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Dual Flash / Programming via RAM

Lab – Flash Programming Tools

- Flash Programming Overview
- FLASH MCU Serial Programmer & Hardware requirements
- USB Direct Programmer & Hardware requirements
- FLASH MCU Serial Walk Through
- USB Direct Walk Through
- Lab Exercises One and Two
- Introduction to other Third Party Programming Partners

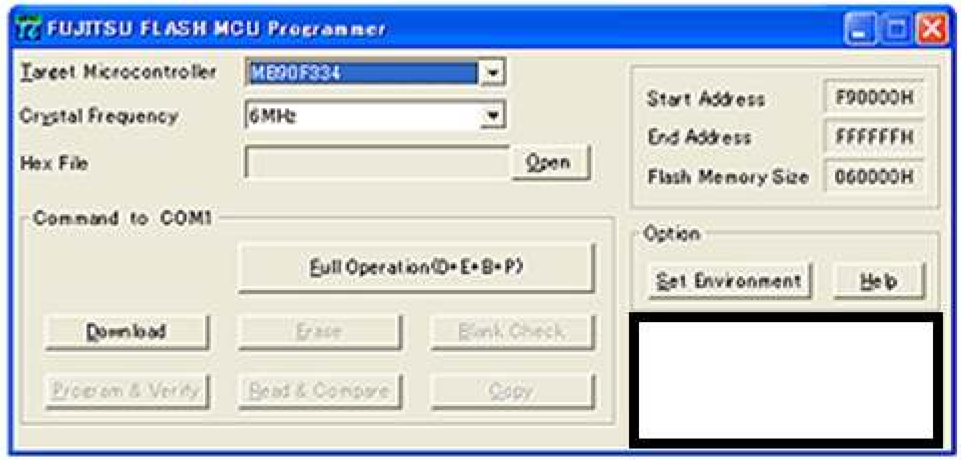
Flash Programming Tools

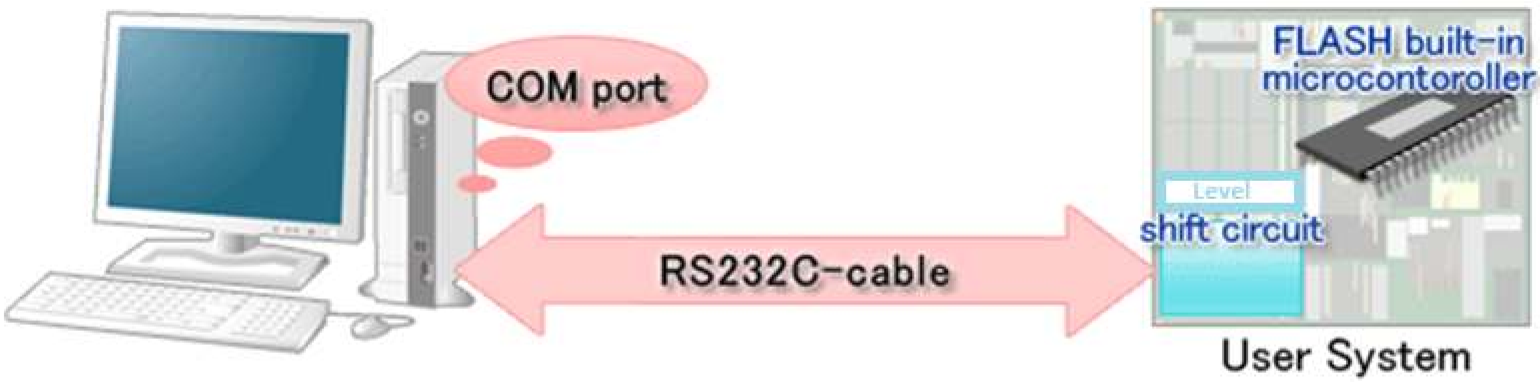
SK-FM4-176L-S6SE2CC Options for Programming Internal Flash

- There are several options to program the microcontroller's flash:
  - JTAG Programming
    - Mostly for debug and small prototype runs. Explained in tool overviews.
  - Flash MCU Serial Programmer - Serial via Serial bridge
    - On CD in \tools\USBDIRECT\PCWFM3.zip
    - After installation USB driver is located in \Program Files (x86)\Spansion\FLASH USB DIRECT Programmer\driver
  - Flash USB DIRECT Programmer via USB device port
    - On CD in \tools\USBDIRECT\USBDIRECT-V01L11\setup.exe
    - After installation USB driver is located in \Program Files (x86)\Spansion\FLASH USB DIRECT Programmer\driver
  - Lots of other third party bulk and production programing options.

FLASH MCU Serial Programmer & Hardware requirements

COM port of PC and target board are connected via RS232C-cable.
On the target board side, the circuit that converts the signal level of RS232C into the signal level of the microcomputer is necessary.

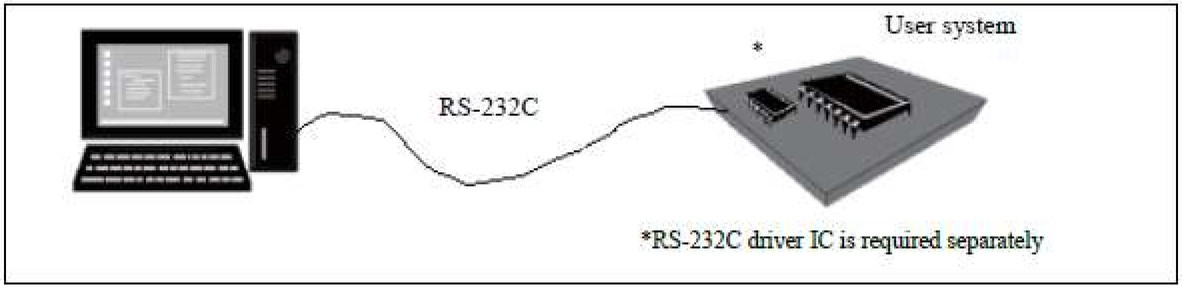




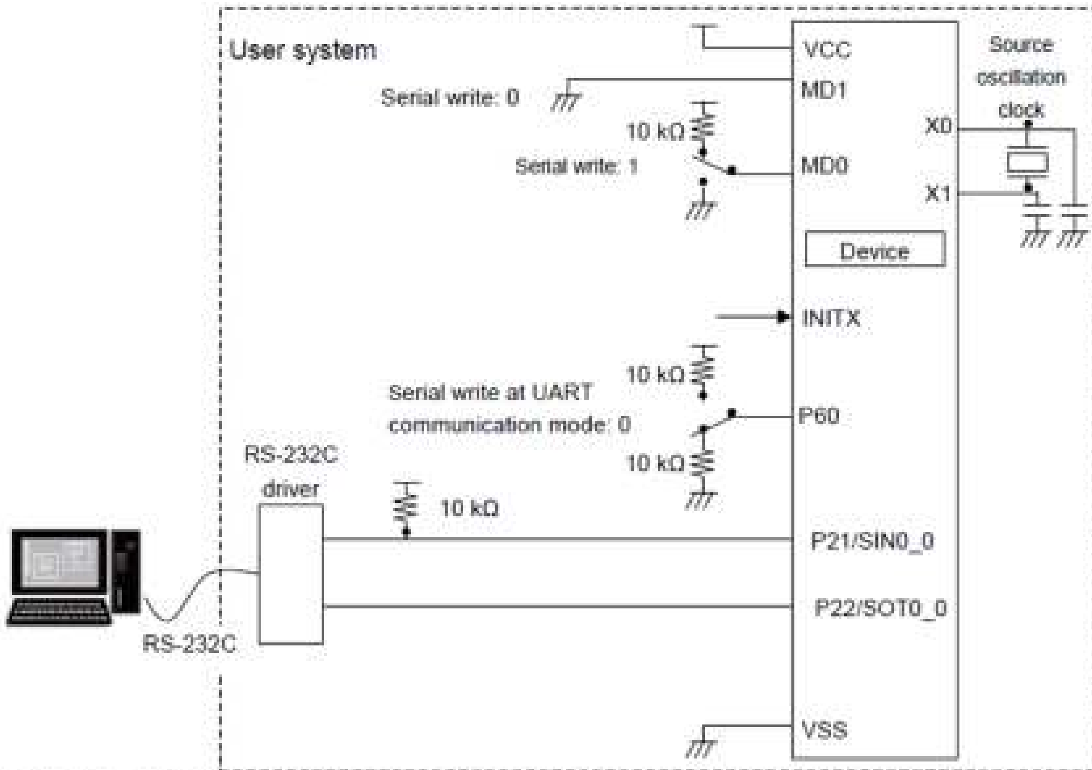
SPANSION

FLASH MCU Serial Programmer & Hardware requirements

Basic Configuration of SPANSION MCU Programmer



Connection Example when Crystal Oscillator is Used



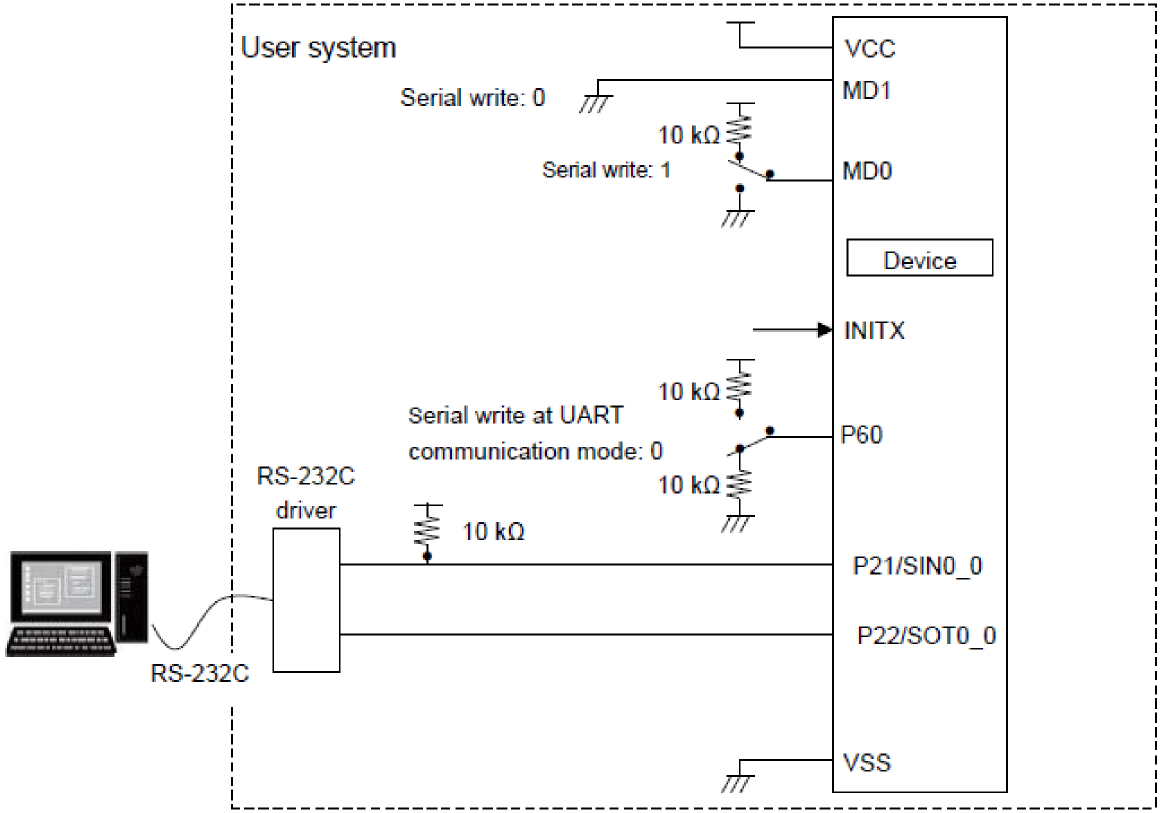
Note: The pull-up resistance values shown are for example. Select the most appropriate resistance values for each system.

FLASH MCU Serial Programmer & Hardware requirements

Oscillator frequency verses communication baud rate

Source Oscillating Frequency	Communication Baud Rate
4MHz	9600bps
8MHz	19200bps
16MHz	38400bps
24MHz	57600bps
48MHz	115200bps

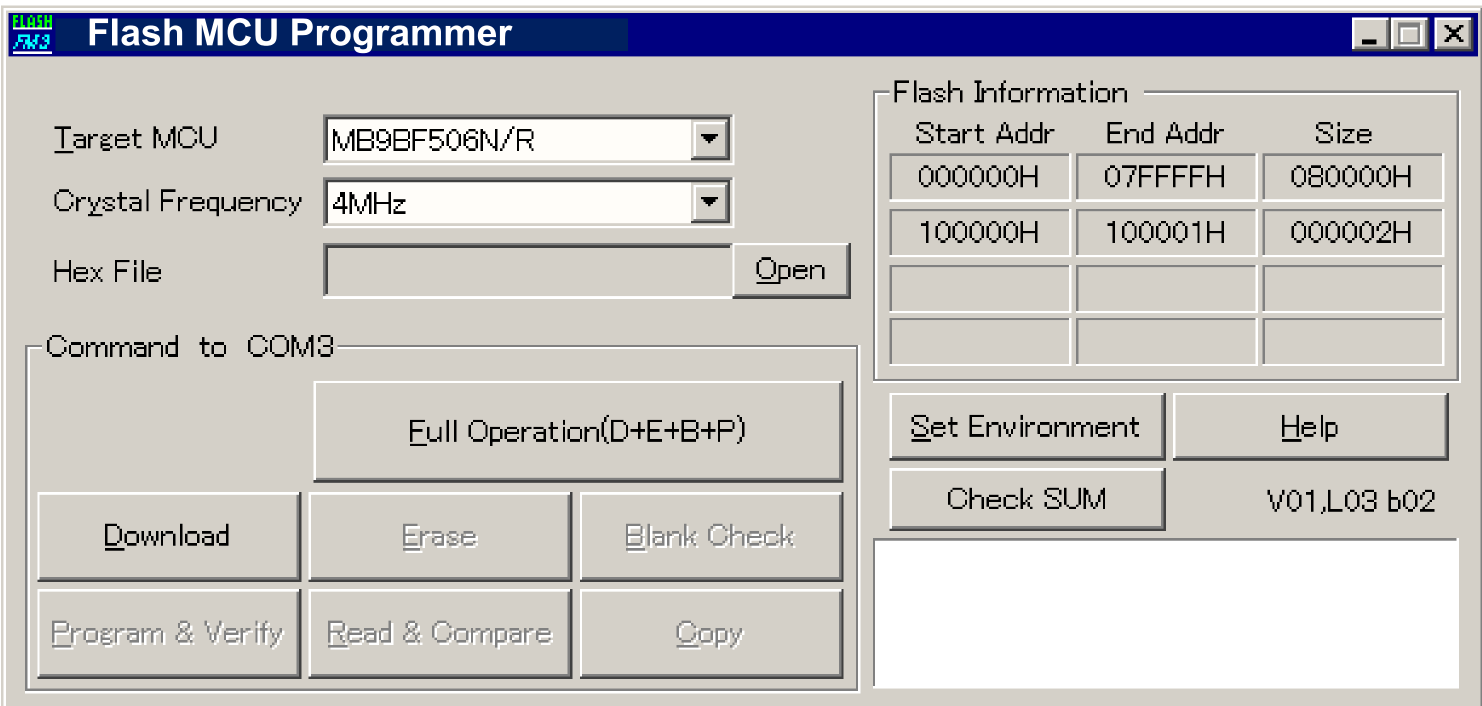
Connection Example When Built-in High-speed CR Oscillator is used



Note: The pull-up resistance values shown are for example. Select the most appropriate resistance values for each system.

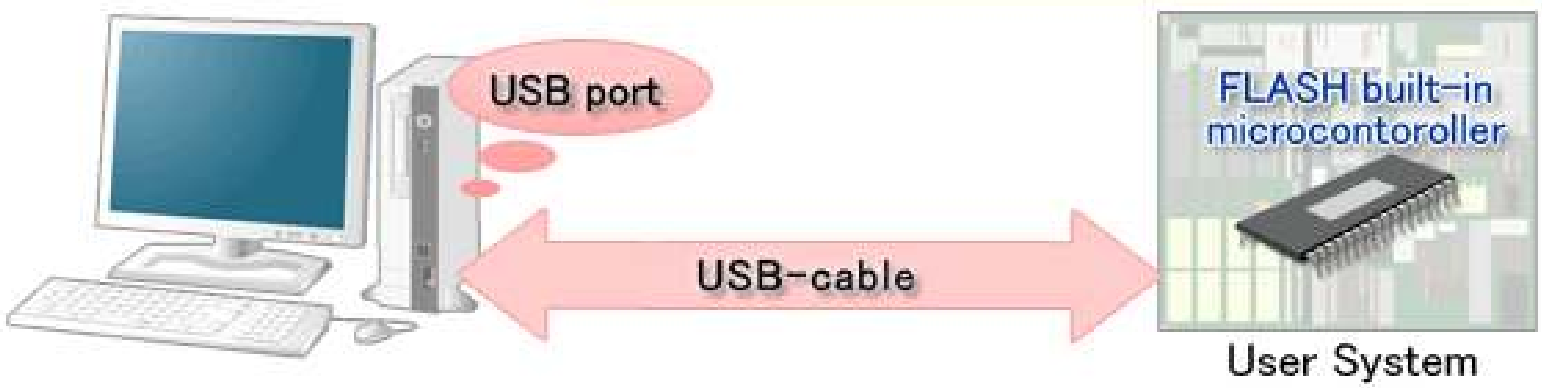
FLASH MCU Serial Programmer & Hardware requirements

- Flash MCU Programmer
  - Free of charge, no registration required
  - Windows based programming tool for Spansion ARM microcontrollers
  - Uses PC serial port COMx (incl. virtual COM port: USB-to-RS232)
    - Serial programming interface for S6SE2CC: UART0



USB Direct Programmer & Hardware requirements

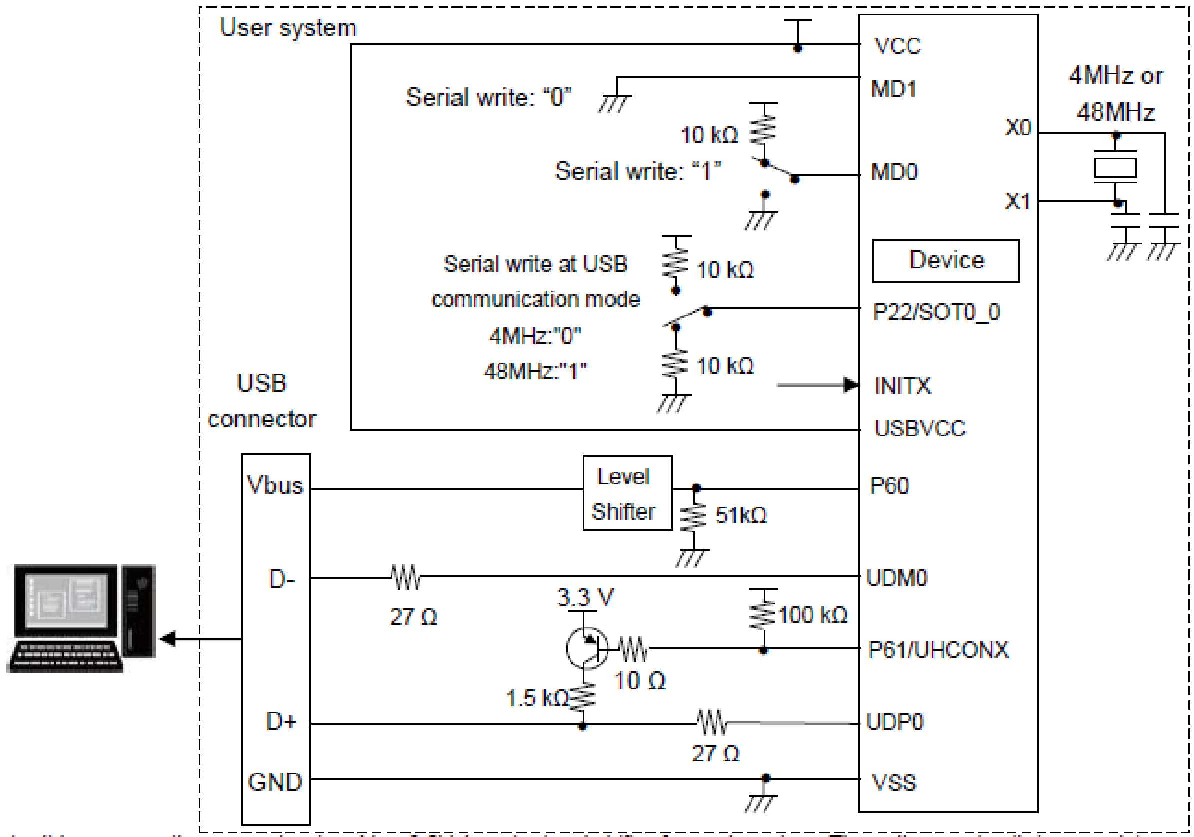
USB port of PC and target board are connected via USB-cable.



SPANSION

Basic Configuration of USB DIRECT Programmer

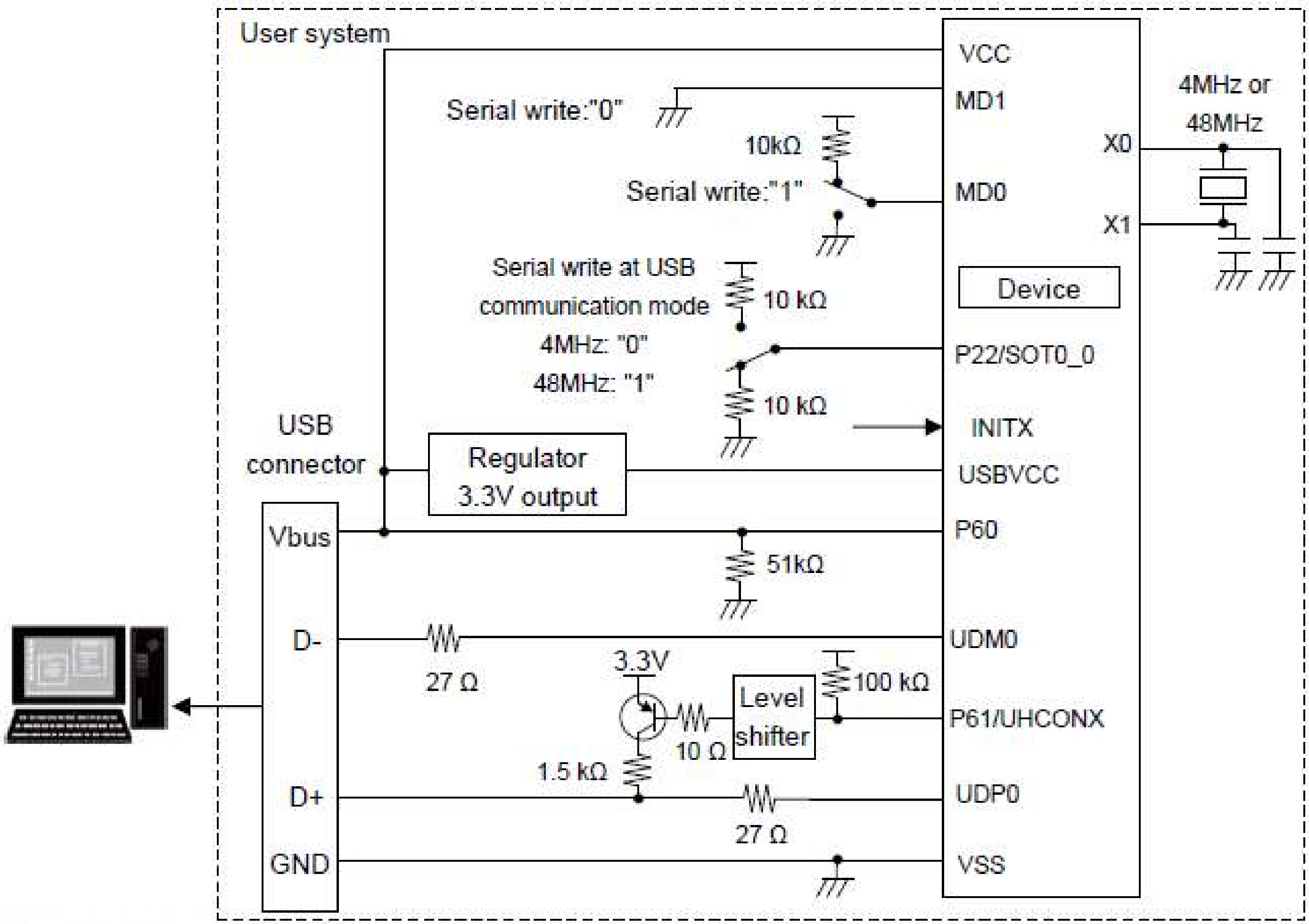
Example using USB DIRECT Programmer (Self Powered.)



Note: It is a connection example when V_{CC}=3.3V. Insert a level shifter for each system. The pull-up and pull-down resistance values shown are for example. Select the most appropriate resistance values for each system.

USB Direct Programmer & Hardware requirements

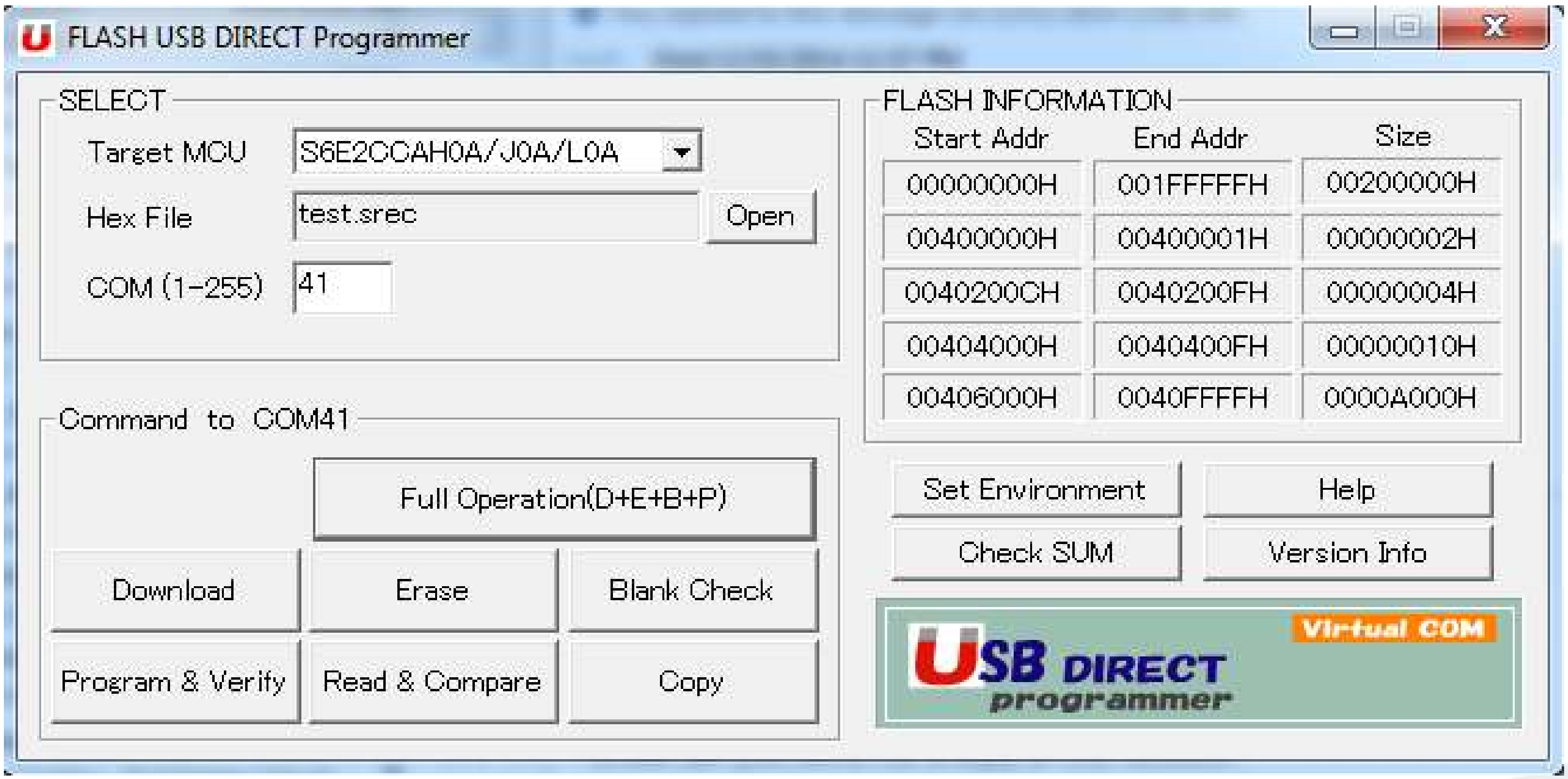
Example using USB DIRECT Programmer (USB Vbus powered.)



Note: The pull-up and pull-down resistance values shown are for example.
Select the most appropriate resistance values for each system.

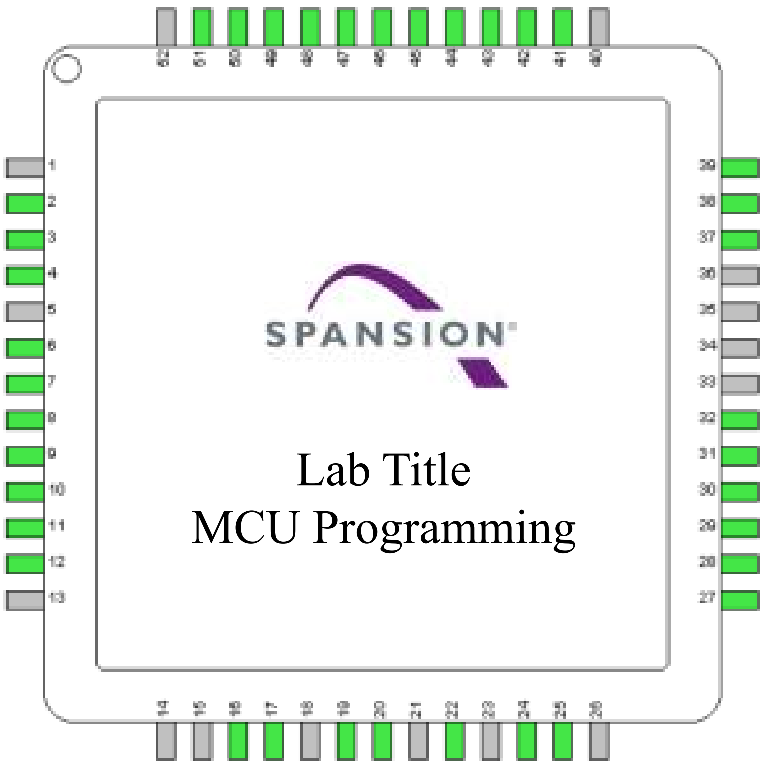
USB Direct Programmer & Hardware requirements

- Direct programming via USB possible
  - Windows-based programming tool for FM3/FM4 microcontroller
    - FM0+ is separate but similar tool



Current version
V01,L11

Lab: SK-FM4-176L-S6SE2CC MCU Serial and USB programming



Description:

Lab will demonstrate programming Spansion MCU by serial or virtual serial port and also by MCU USB port

Objective:

To learn MCU Flash programming in order to be able to extract customer code from programmed device.

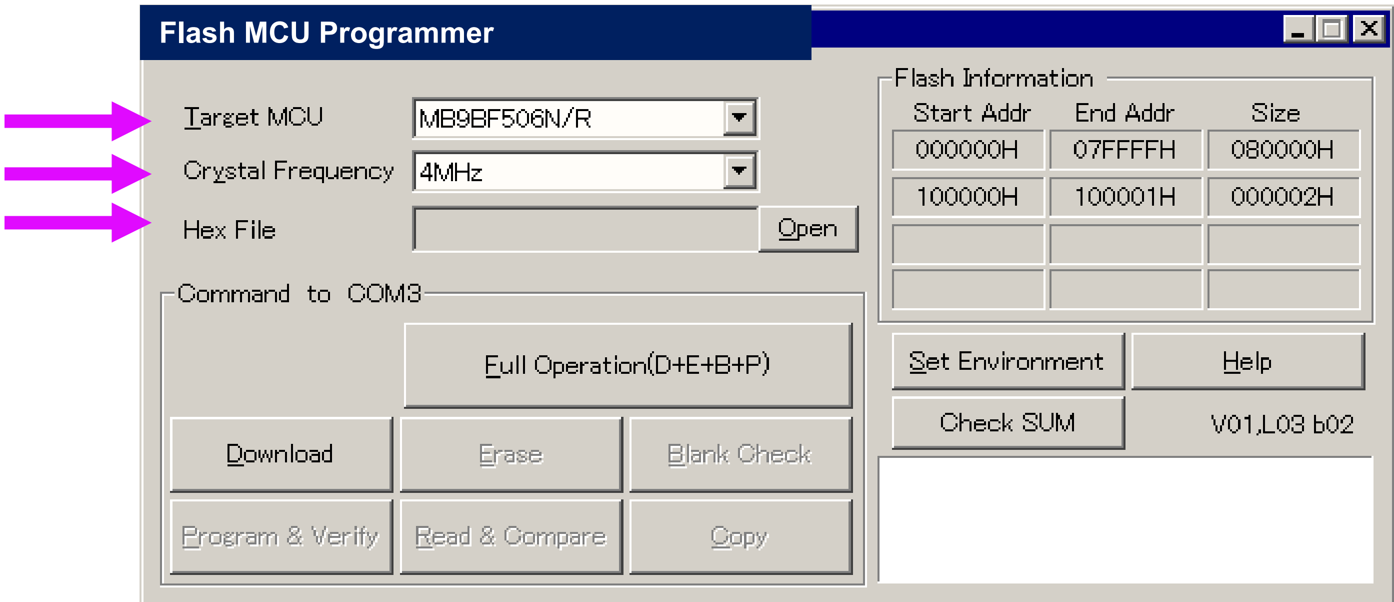
To learn to program demo or example code into an MCU in a quick and efficient manner to show items without having to use the whole IDE product.

1st Exercise: Flash MCU Serial Programmer

- FLASH Serial Programming via CN2 (UART0 input to MCU)
  - **Configure the SK target for programming**
    - Select the MCU power supply (J4 - DAP)
    - Install J2 (BiROM PROG)
    - Connect USB port from PC to CMSIS DAP USB CN2 Virtual COM port (CN2)
    - If connected for first time, Windows OS may ask for a driver
      - See "C:\Program Files (x86)\Spansion\FLASH MCU Programmer\FM3\flash.exe" (default location)
  - **Start the FLASH MCU Serial Programmer**
    - If required install :
      - s6e2cc_flashprogramming_v10.zip\Flash Programming\PCWFM3.zip
    - Start flash.exe
      - C:\Program Files (x86)\Spansion\FLASH MCU Programmer\FM3\flash.exe
    - Select the COM port and download srec file
    - Press Reset
    - Start Full Operation
    - After all the dialog boxes clear, programming is complete.

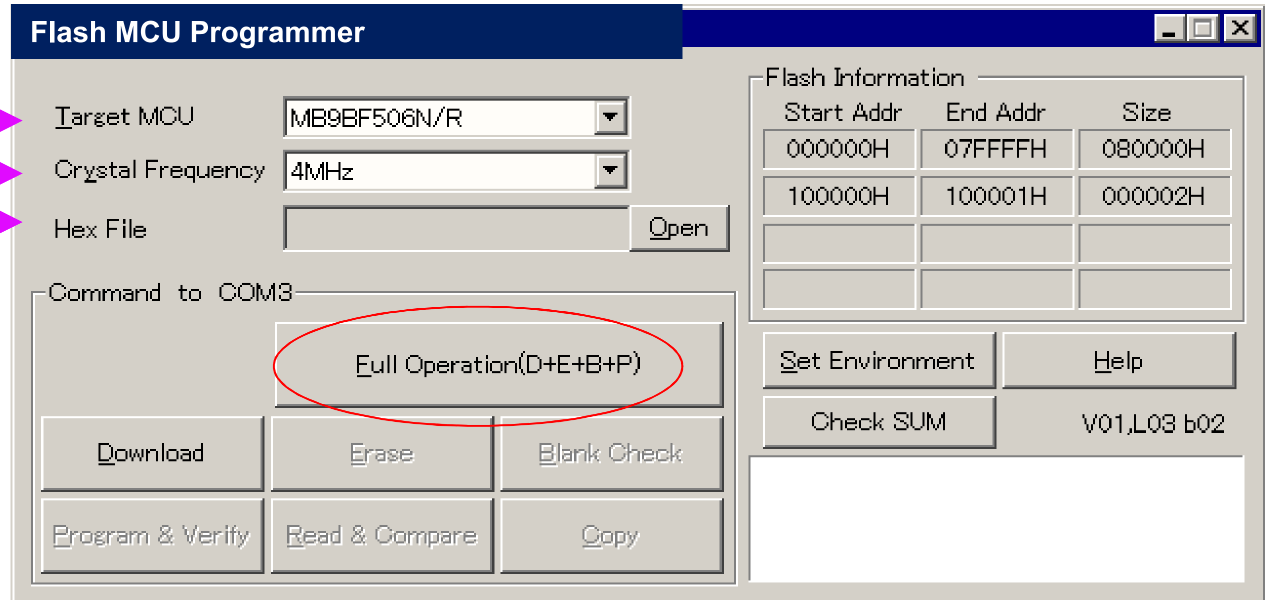
1st Exercise: Flash MCU Serial Programmer

- Start the MCU Flash Programmer
- Select the *target microcontroller* (S6SE2CC)
- Select the *crystal frequency* (4 MHz)
- Choose the software srec:
  - **s6e2cc_flashprogramming_v10.zip\Flash Programming\test.srec**
  - **s6e2cc_flashprogramming_v10.zip\Flash Programming\blink.srec**

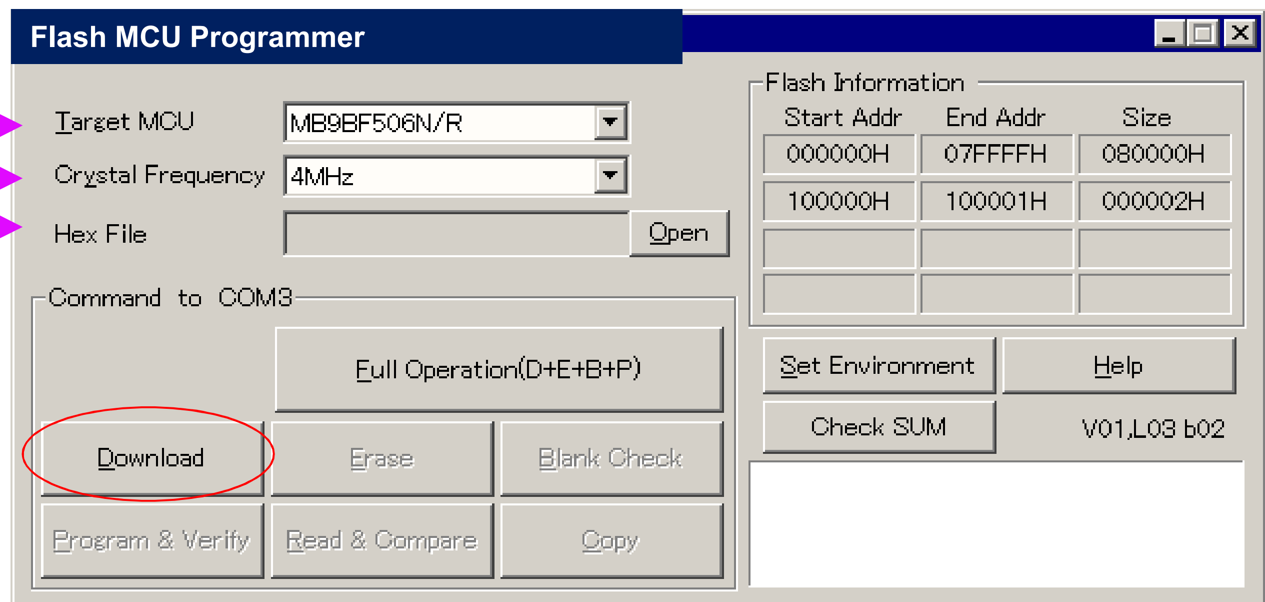


1st Exercise: Flash MCU Serial Programmer

The Full Operation button will take the selected File and program it into the device.



To enable reading of flash content or mass erase of Flash use the Download (kernel) button



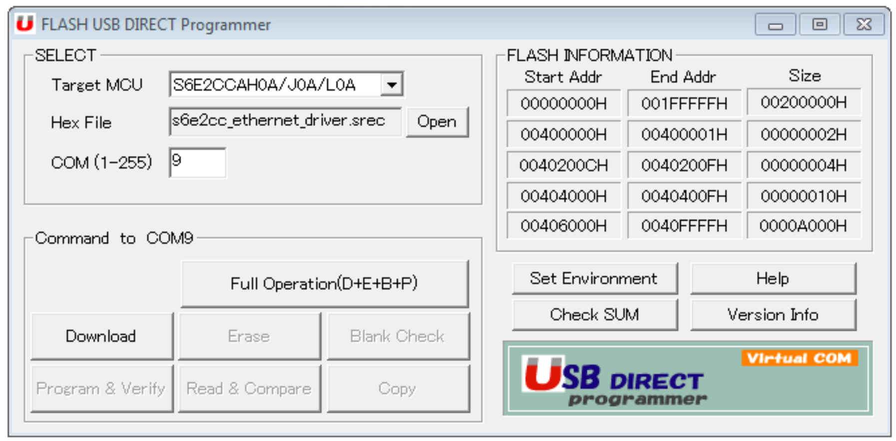
2nd Exercise: Flash Programming via MCU USB (DEVICE MODE)

- FLASH USB DIRECT Programming via CN3 (USB input to MCU)
  - **Configure the SK target for programming**
    - Select the MCU power supply (J4 - USB)
    - Install J2 (USB PROG)
    - Connect USB port CN3 with the PC (not CMSIS DAP USB CN2)
    - If connected for first time, Windows OS may ask for a driver
      - See C:\Program Files (x86)\Spansion\FLASH USB DIRECT Programmer\driver (default location)
  - **Start the FLASH USB DIRECT Programmer**
    - Start flash.exe
      - See: C:\Program Files (x86)\Spansion\FLASH USB DIRECT Programmer\flash.exe
    - For first installation, Windows OS may ask for a driver
      - See C:\Program Files (x86)\Spansion\FLASH USB DIRECT Programmer\driver (default location)
    - Select the COM port and download file
    - Press Reset
    - Start Full Operation
    - After all the dialog boxes clear, programming is complete.

2nd Exercise: Flash Programming via MCU USB (DEVICE MODE)

- Select the correct target MCU: S6E2CCAH0A/J0A/L0A
- Browse for the programming file (*.srec or *.hex)
  - s6e2cc_flashprogramming_v10.zip\Flash Programming\test.srec
  - s6e2cc_flashprogramming_v10.zip\Flash Programming\blink.srec
- Adjust the corresponding virtual COM-port

Select MCU: S6E2CCAH0A/J0A/L0A
Select file (*.srec; *.hex)
Select Virtual COM-port



- Use -> Full Operation
  - Download kernel
  - Erase flash memory / Blank check
  - Program & Verify project to flash memory
- Remove program shunt on J2, move USB cable back to CN2, place power select shunt @ J4 back to DAP, and press Reset button.
- **New program is now running**

Lab: SK-FM4-176L-S6SE2CC MCU Serial and USB programming

Remember:

Select the MCU correct power supply (J4 = USB / J4 = DAP), Install J2 = BiROM PROG

Connect USB port CN3 with the PC for USB direct.

Connect PC to CN2 CMSIS DAP Virtual serial port for Flash MCU Serial Programmer

Select the COM port and download file, Select Start Full Operation, follow as directed

----

- Load s6e2cc_flashprogramming_v10.zip\Flash Programming\blink.srec – Once run, your should be able to press the user switch and the RGB LED will cycle colors
- Reload s6e2cc_flashprogramming_v10.zip\Flash Programming\test.srec – This is the original production code that was in the board out of the box. Remember to set the board back to its original configuration and connect aTerminal at 115200.

Third party flash programmer solutions.

http://www.spansion.com/Products/microcontrollers/32-bit-ARM-Core/fm3/Pages/Spanion FM MCU Ecosystem.aspx

FLASH Programmer

	IDE/Compiler	Debugger	OS	Middleware	FLASH Programmer	IC Programming Service	Evaluation Board	Simulator	Training	Engineering Service
Computex	✓	✓			✓					
CONITEC					✓					
CooCox	✓	✓	✓	✓	✓					
ELNEC					✓					
EMPROG	✓	✓	✓	✓	✓		✓		✓	
Falcon Electronics					✓	✓				
Flash Support Group					✓					
Hitex		✓	✓	✓	✓					
MINATO ELECTRONICS					✓	✓				
NAITO DENSEI MATCHIDA MFG					✓					
Rowley	✓	✓	✓		✓			✓		
SEGGER		✓	✓	✓	✓					
Sohwa & Sophia Technologies	✓	✓	✓	✓	✓		✓	✓	✓	
Yokogawa Digital Computer		✓			✓				✓	
Wave Technology					✓					
XELTEK					✓	✓				
	IDE/Compiler	Debugger	OS	Middleware	FLASH Programmer	IC Programming Service	Evaluation Board	Simulator	Training	Engineering Service

SPANSION

Flash Programming Tools

What just happened?

- We briefly discussed Flash Programming Overview
- Introduced the hardware needed to implement different programming options
- Introduction to Flash MCU Programmer.
- Introduction to USB Direct Programmer.
- Used tools to load a couple of srec files into target board to prove functionality
- Introduced other Third Party Programming Partners
- More information:
  - http://www.spansion.com/Support/microcontrollers/developmentenvironment/Pages/index.aspx

SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Quad SPI flash interface
- Lab - Dual Flash / Programming via RAM

SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Quad SPI flash interface
- Lab - Dual Flash / Programming via RAM

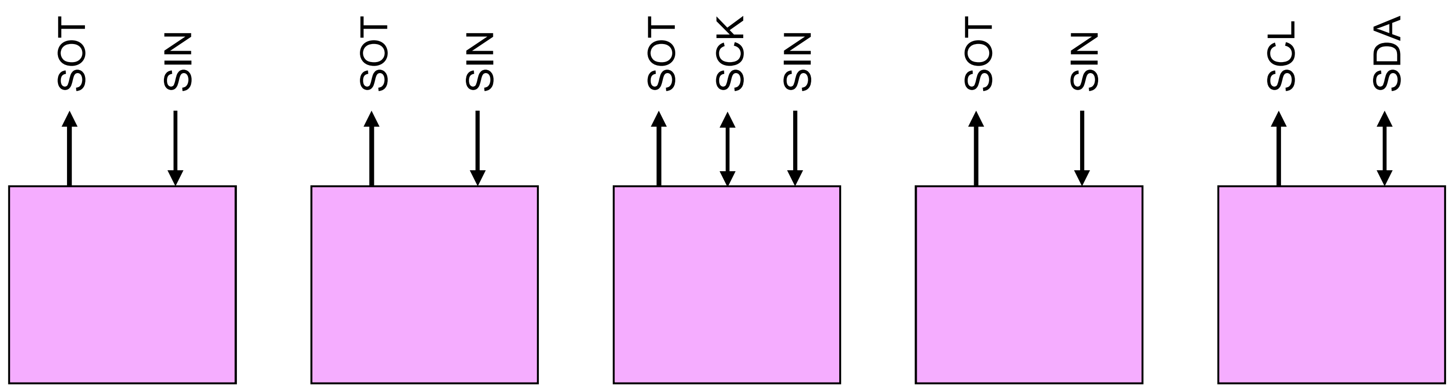
Lab - MCU Template UART -> C printf

Introduction to Multi-Function Serial (MFS)
Lab using MCU template UART output
Lab using higher-level printf

Lab - MCU Template UART -> C printf

Multifunctional Serial Interface

Mode Bits of SMR	Mode 0	Mode 1	Mode 2	Mode 3	Mode 4
	Asynchronous normal	**Asynchronous multi-processor**	**Synchronous**	LIN	I^2C
MD2	0	0	0	0	1
MD1	0	0	1	1	0
MD0	0	1	0	1	0



Lab - MCU Template UART -> C printf

Multifunctional Serial Interface

- Dedicated reload baud rate counter
  - 15-bit reload counter for adjusting baud rate
  - Clockable internally and externally
- Data format
  - NRZ, inverted NRZ
  - LSB first, MSB first
- Data length
  - 5-9 bits in asynchronous normal mode w/wo additional parity bit
  - 5-16 bits in synchronous mode (including SPI)
  - 7-8 bits in asynchronous multi-processor mode
- Error detection
  - Framing error, Overrun error, Parity error

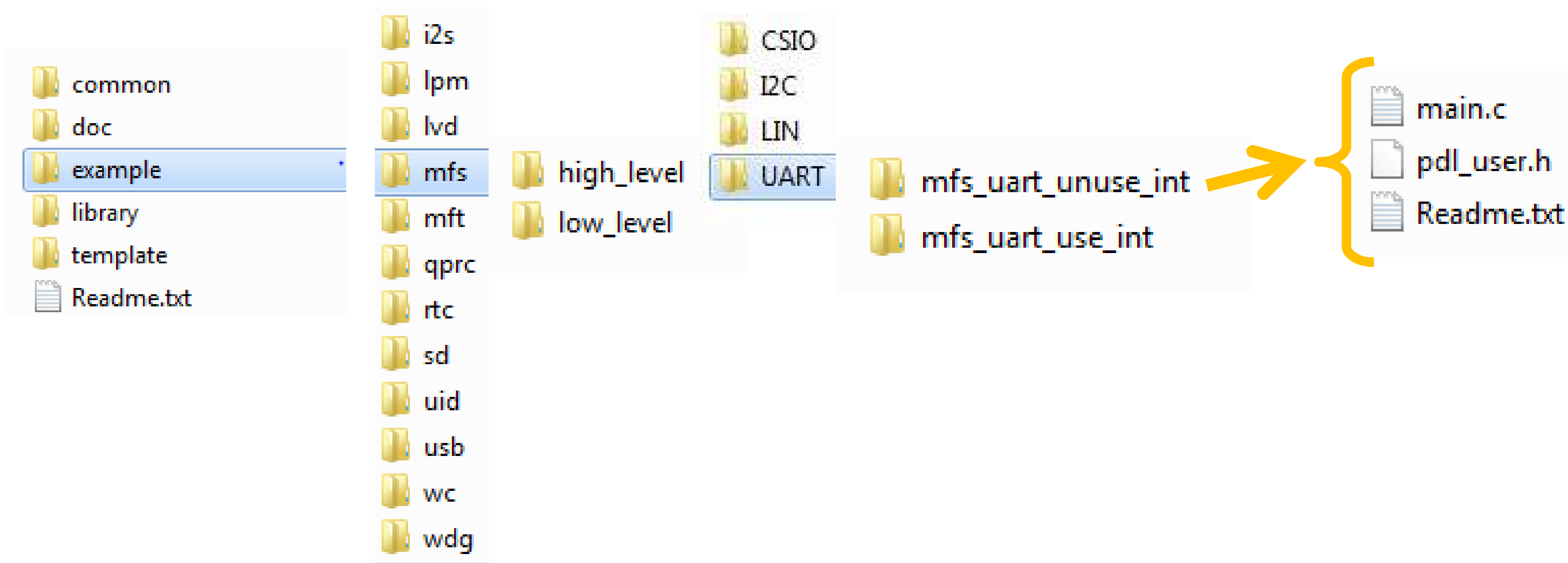
Lab - MCU Template UART -> C printf

Multifunctional Serial Interface

- Interrupt control
  - Reception interrupt
    - Handles also error bits
  - Transmission interrupt
    - Handles also status interrupt (Lin Break Detection)
- FIFO
  - Reception FIFO (64 bytes)
  - Transmission FIFO (64 bytes)
- Hardware flow control
  - Available on MFS4, MFS5
  - Handshaking signals
    - CTSx signal output
    - RTSx signal input
- Hardware programable chip select signal (synchronous mode)
  - Available on MFS6, MFS7

Lab - MCU Template UART -> C printf

Remember PDL Example Folders:
Labs\Master PDL\S6E2CC_PDL v0.2\example\mfs\high_level\UART\mfs_uart_unuse_int

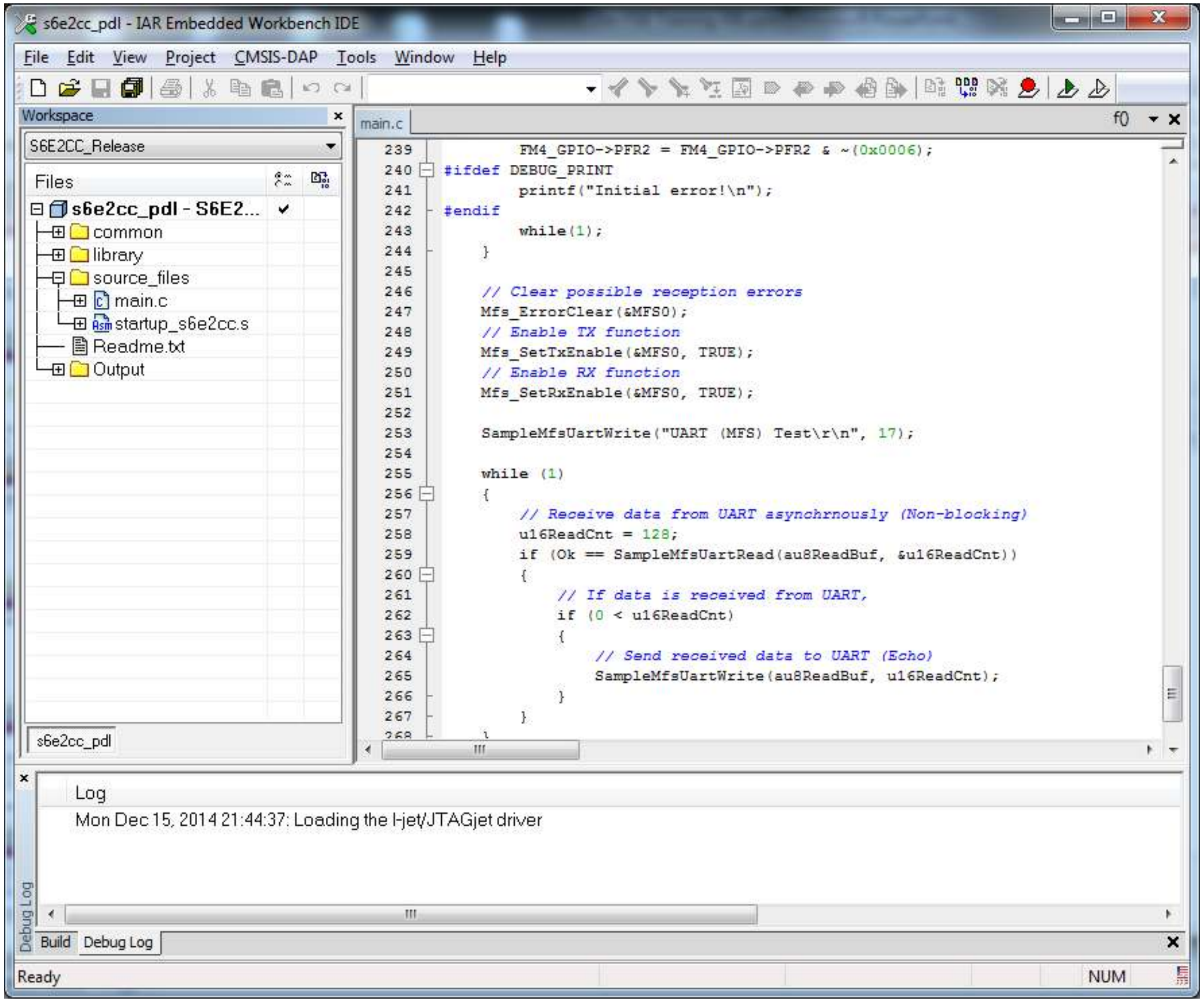


Go to the examples from the PDL as shown above. Take the three files and over-write the files in the source directory:

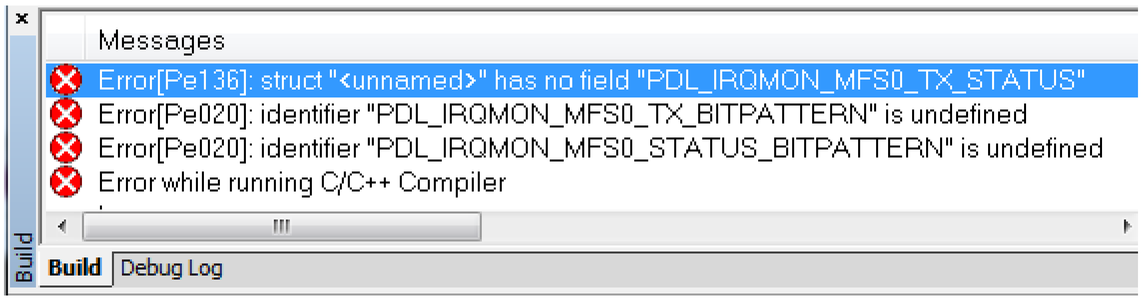
LABs\s6e2cc_PDL_v0.2\S6E2CC_PDL v0.2\template\source\
… main.c, pdl_user.h, readme.txt

Lab - MCU Template UART -> C printf

Launch IAR and re-open the workspace at:
Labs\Master PDL\s6e2cc_PDL_v0.2\template\IAR\s6e2cc_pdl.eww



From Project pull-down select rebuild all and then Launch the debugger with the green  **GO button.**



- Note – Large RED X's are a bad thing. Debugger will not launch with complier errors. What do we do now? Suggestions?
- Are we sure a Rebuild All was done? Remember that we switch files on the tool. Not doing a rebuild all leaves the tool in a sick state with half and half code.
- Try it again and make sure it is REBUILD ALL

Lab using MCU template UART

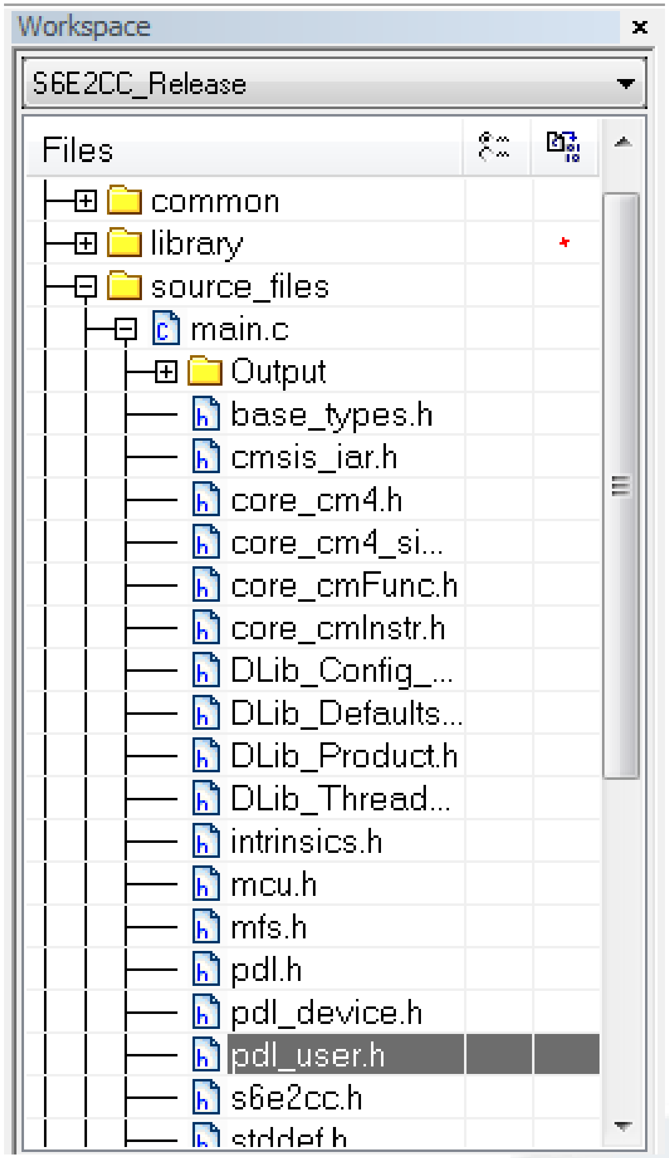
As can be seen below it appears that there is a more serious problem.
Look at the routine that has the problems (below) what can you say about it?

Two things should give a clue. It appears to be an ISR and there are a lot of PDL_ON / PDL_OFF defines all around it

```
 ******************************************************************************
 ** \brief MFS ch. 0 TX/Status IRQ handler
 ******************************************************************************
 */
void MFS0_TX_IRQHandler(void)
{
  #if (PDL_ON == PDL_DSTC_ENABLE_MFS0_TX)
    Dstc_MfsRxIrqHandler(DSTC_IRQ_NUMBER_MFS0_TX);
  #else
    uint32_t u32IrqMon = FM4_INTREQ->PDL_IRQMON_MFS0_TX_STATUS;

    if((u32IrqMon & (PDL_IRQMON_MFS0_TX_BITPATTERN)) == PDL_IRQMON_MFS0_TX_BITPATTERN)
    {
        MfsIrqHandlerTx((stc_mfsn_t*)&MFS0, &(m_astcMfsInstanceDataLut[MfsInstanceIndexMfs0].stcInternData));
    }

    if((u32IrqMon & (PDL_IRQMON_MFS0_STATUS_BITPATTERN)) == PDL_IRQMON_MFS0_STATUS_BITPATTERN)
    {
        MfsIrqHandlerStatus((stc_mfsn_t*)&MFS0, &(m_astcMfsInstanceDataLut[MfsInstanceIndexMfs0].stcInternData));
    }
  #endif
}
#endif // #if (PDL_INTERRUPT_ENABLE_MFS0 == PDL_ON) && (PDL_PERIPHERAL_ENABLE_MFS0 == PDL_ON)
```

Lab - MCU Template UART -> C printf

- Since there were only two files in this project let's look at the other one.
- Open PDL_user.h



  CONFIDENTIAL

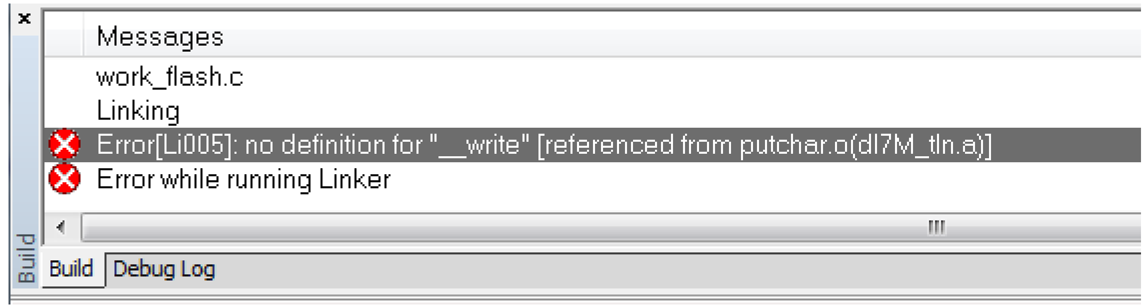
Lab - MCU Template UART -> C printf

- A quick scan of enabled and disabled modules, does not show anything unusual the MFS0 is about all that is turned on.
- Look further down and we see that the MFS0 interrupt is turned on. Wasn't the project called "mfs_uart_unuse_int"?
- Turn the define to PDL_OFF and do a rebuild all

```c
// External Bus Interface
#define PDL_INTERRUPT_ENABLE_EXTIF                PDL_OFF

// HDMI routines
#define PDL_INTERRUPT_ENABLE_HDMI                 PDL_OFF

// I2S
#define PDL_INTERRUPT_ENABLE_I2S0                 PDL_OFF

// I2S Clock
#define PDL_INTERRUPT_ENABLE_I2S_CLK              PDL_OFF


// LCD
#define PDL_INTERRUPT_ENABLE_LCD                  PDL_OFF

// Low Voltage Detection
#define PDL_INTERRUPT_ENABLE_LVD                  PDL_OFF

// Multi Function Serial Interfaces
#define PDL_INTERRUPT_ENABLE_MFS0                 PDL_ON
#define PDL_INTERRUPT_ENABLE_MFS1                 PDL_OFF
#define PDL_INTERRUPT_ENABLE_MFS2                 PDL_OFF
#define PDL_INTERRUPT_ENABLE_MFS3                 PDL_OFF
```

Lab - MCU Template UART -> C printf

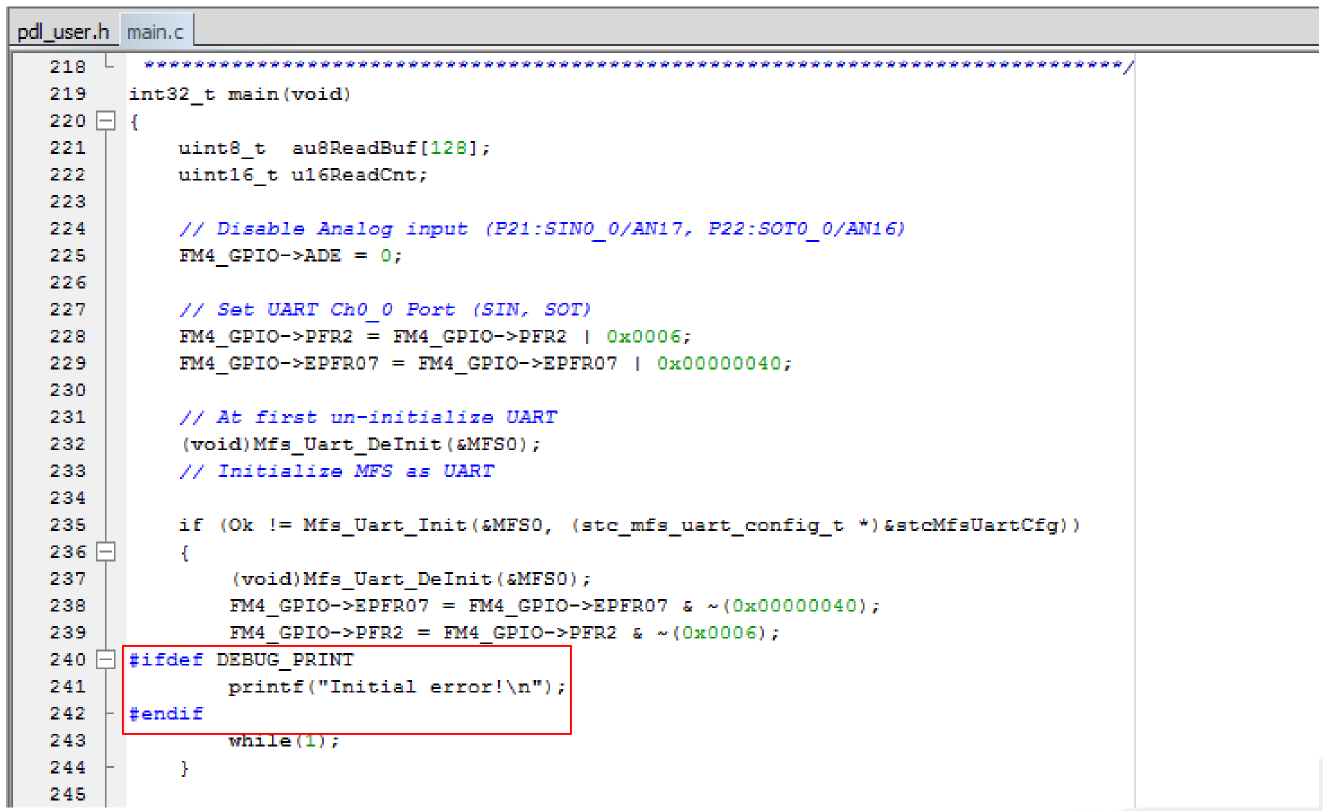
How embarrassing. It is still not compiling. The last error message is referring to a problem in the putchar.c file.



Scanning through main in main.c notice that there is a # define with a printf in it. This looks suspicious.

Just use // on lines 240, 241, 242 and get rid of it.

Try the Rebuild All one last time before we give up and call factory applications



pdl_user.h main.c

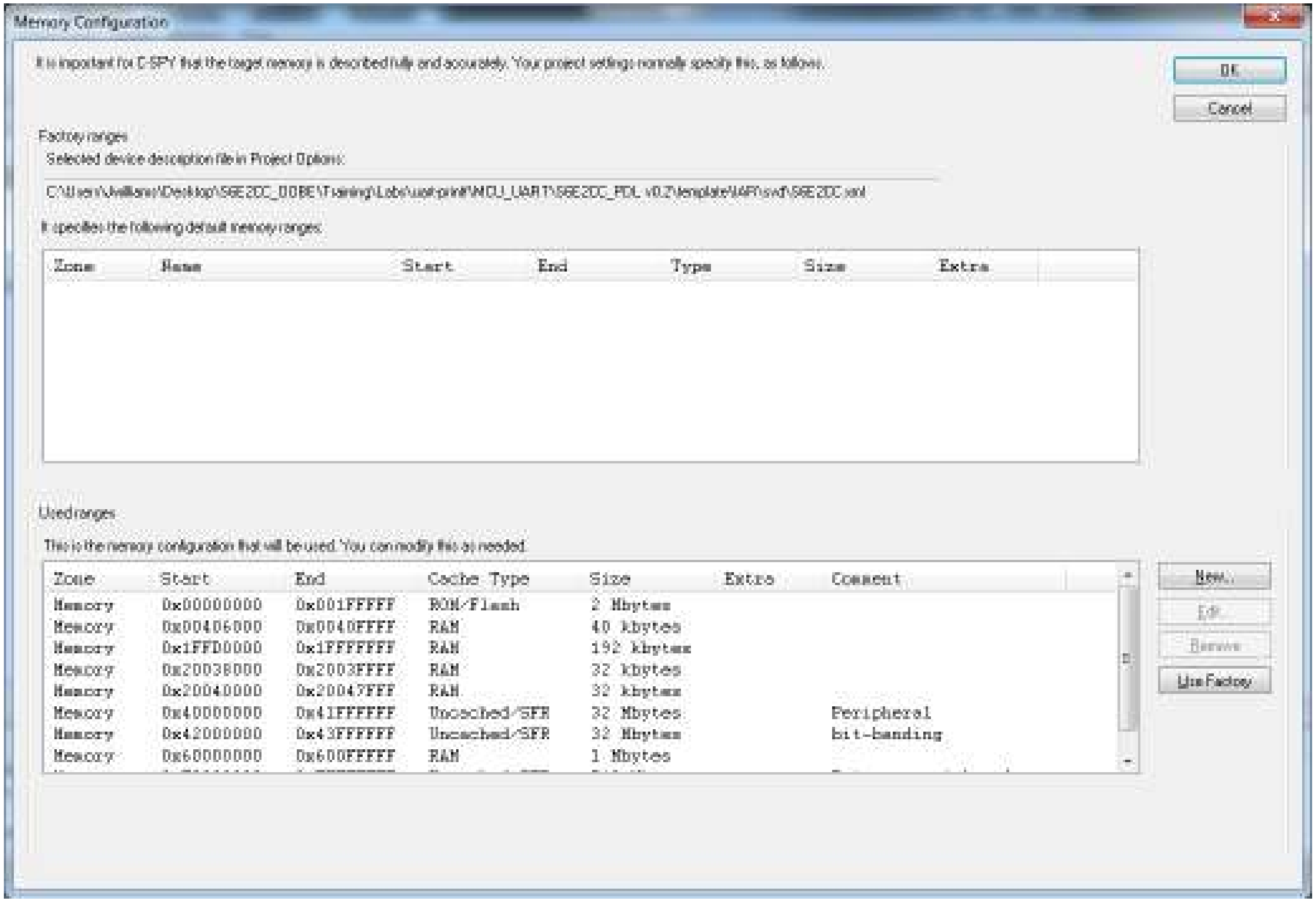
```
218  ******************************************************************************/
219  int32_t main(void)
220  {
221      uint8_t  au8ReadBuf[128];
222      uint16_t u16ReadCnt;
223
224      // Disable Analog input (P21:SIN0_0/AN17, P22:SOT0_0/AN16)
225      FM4_GPIO->ADE = 0;
226
227      // Set UART Ch0_0 Port (SIN, SOT)
228      FM4_GPIO->PFR2 = FM4_GPIO->PFR2 | 0x0006;
229      FM4_GPIO->EPFR07 = FM4_GPIO->EPFR07 | 0x00000040;
230
231      // At first un-initialize UART
232      (void)Mfs_Uart_DeInit(&MFS0);
233      // Initialize MFS as UART
234
235      if (Ok != Mfs_Uart_Init(&MFS0, (stc_mfs_uart_config_t *)&stcMfsUartCfg))
236      {
237          (void)Mfs_Uart_DeInit(&MFS0);
238          FM4_GPIO->EPFR07 = FM4_GPIO->EPFR07 & ~(0x00000040);
239          FM4_GPIO->PFR2 = FM4_GPIO->PFR2 & ~(0x0006);
240  #ifdef DEBUG_PRINT
241          printf("Initial error!\n");
242  #endif
243          while(1);
244      }
245
```


Lab - MCU Template UART -> C printf

In IAR, select project from the title menu and then click rebuild all.
If build is successful launch the debugger with the  button in title bar.

In some cases you may get a popup to asking you to acknowledge the project memory map. - Except for a few special cases just click OK.

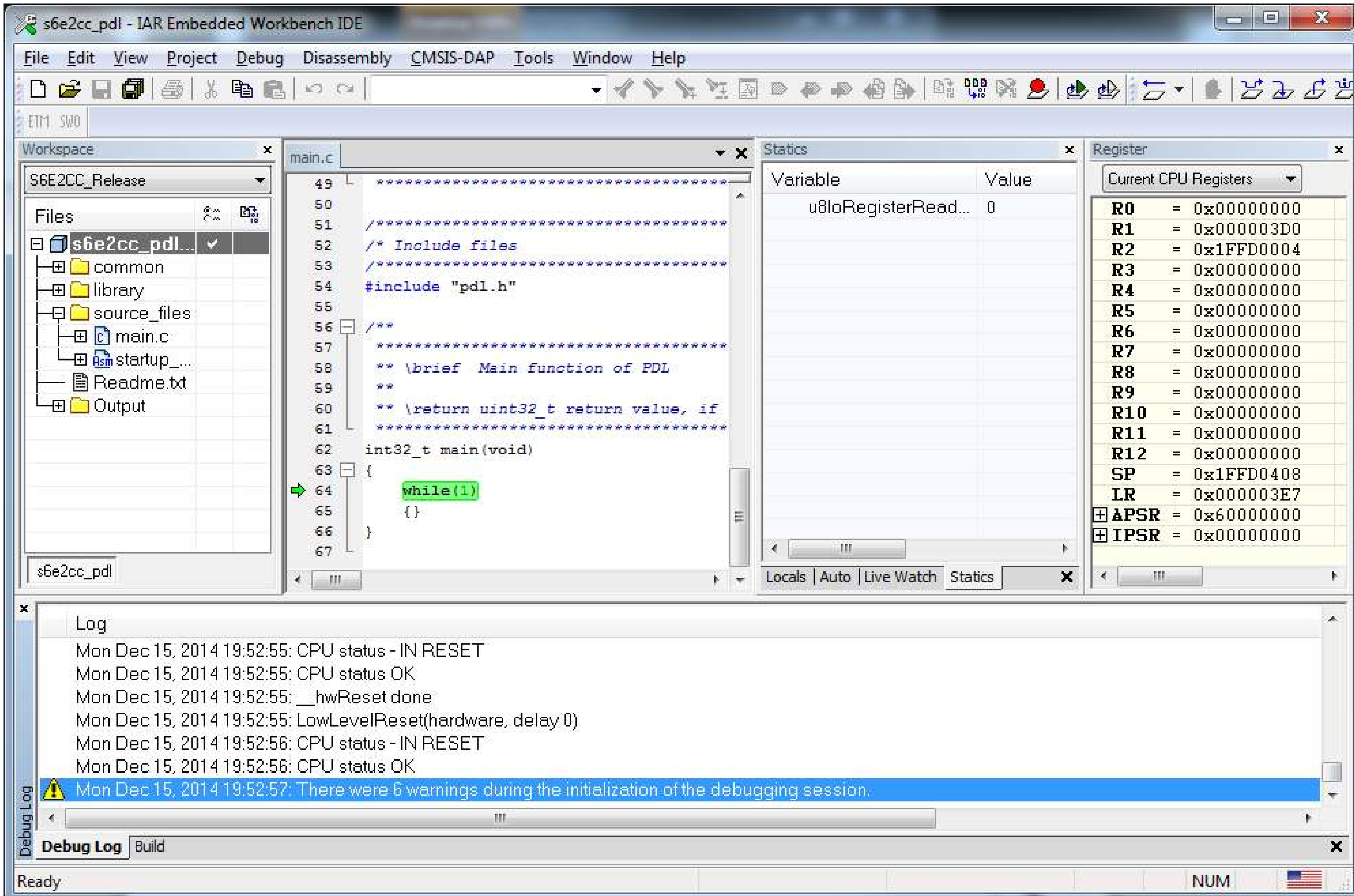
Then the memory map selected will display – OK to continue to the Debugger



  CONFIDENTIAL

Lab - MCU Template UART -> C printf

If things are good at this point the debugger should be paused at the main() loop function.



Lab - MCU Template UART -> C printf

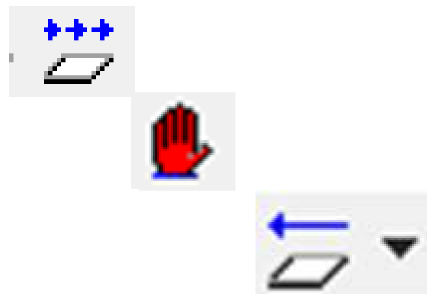
Click the (GO) button and the debugger will start the code running.

Register

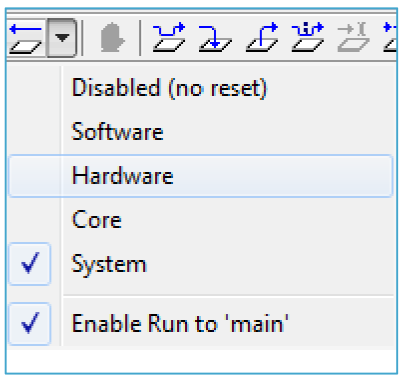
Current CPU Registers

R0	=	0x00000000
R1	=	0x000003D0
R2	=	0x1FFD0004
R3	=	0x00000000
R4	=	0x00000000
R5	=	0x00000000
R6	=	0x00000000
R7	=	0x00000000
R8	=	0x00000000
R9	=	0x00000000
R10	=	0x00000000
R11	=	0x00000000
R12	=	0x00000000
SP	=	0x1FFD0408
LR	=	0x000003E7
⊞APSR	=	0x60000000
⊞IPSR	=	0x00000000
⊞EPSR	=	0x01000000
PC	=	0x000003EE
⊞PRIMASK	=	0x00000000
⊞BASEPRI	=	0x00000000
⊞BASEPRI_MAX	=	0x00000000
⊞FAULTMASK	=	0x00000000
⊞CONTROL	=	0x00000000
CYCLECOUNTER	=	132055

Experiment with the GO, PAUSE, and RESET buttons and note the effect on the IAR CYCLECOUNTER parameter



What gives with the little arrow by reset button?



Register

Current CPU Registers

R0	=	0x00000000
R1	=	0x000003D0
R2	=	0x1FFD0004
R3	=	0x00000000
R4	=	0x00000000
R5	=	0x00000000
R6	=	0x00000000
R7	=	0x00000000
R8	=	0x00000000
R9	=	0x00000000
R10	=	0x00000000
R11	=	0x00000000
R12	=	0x00000000
SP	=	0x1FFD0408
LR	=	0x000003E7
⊞APSR	=	0x60000000
⊞IPSR	=	0x00000000
⊞EPSR	=	0x01000000
PC	=	0x000003EE
⊞PRIMASK	=	0x00000000
⊞BASEPRI	=	0x00000000
⊞BASEPRI_MAX	=	0x00000000
⊞FAULTMASK	=	0x00000000
⊞CONTROL	=	0x00000000
CYCLECOUNTER	=	503561894

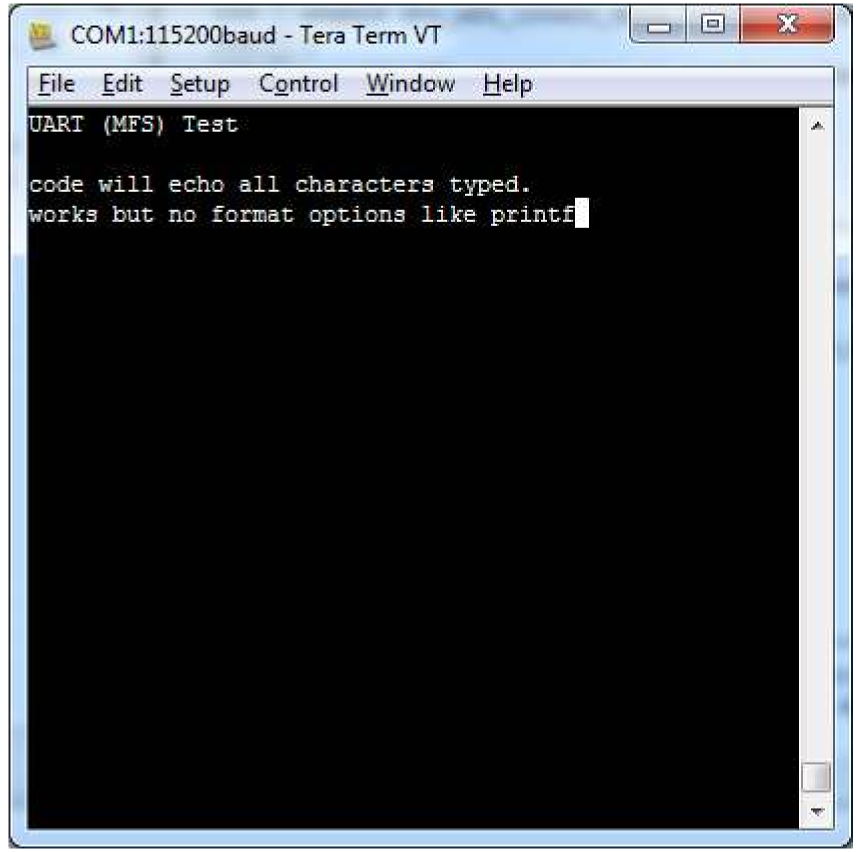
Used to set the type of reset used, and whether to run through sys_init() to main()

Lab using MCU template UART

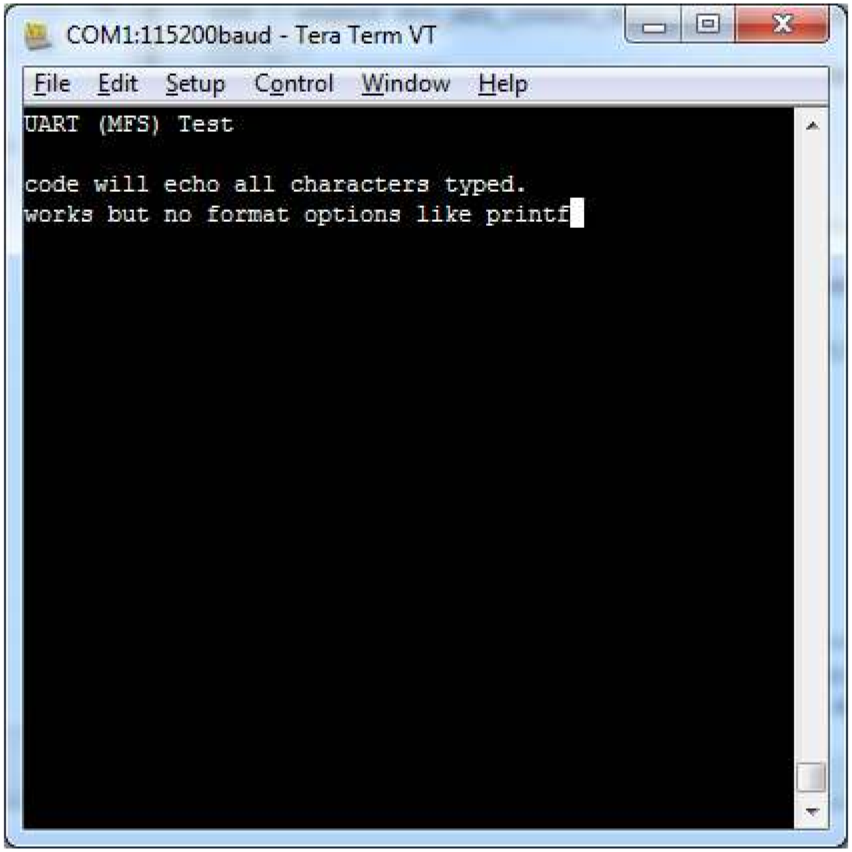
Eureka !! Now open a terminal emulator and launch the debugger and let's see our output.

Lot of work for one line of printed characters, and echo back of typed characters

Since we went to all the effort let's look at some of the features of this project.



UART example Features



Very simple structure and functions:

```
// Clear possible reception errors
Mfs_ErrorClear(&MFS0);
// Enable TX function
Mfs_SetTxEnable(&MFS0, TRUE);
// Enable RX function
Mfs_SetRxEnable(&MFS0, TRUE);

SampleMfsUartWrite("UART (MFS) Test\r\n", 17);
```

This method is very efficient in code and speed, but difficult to format and use

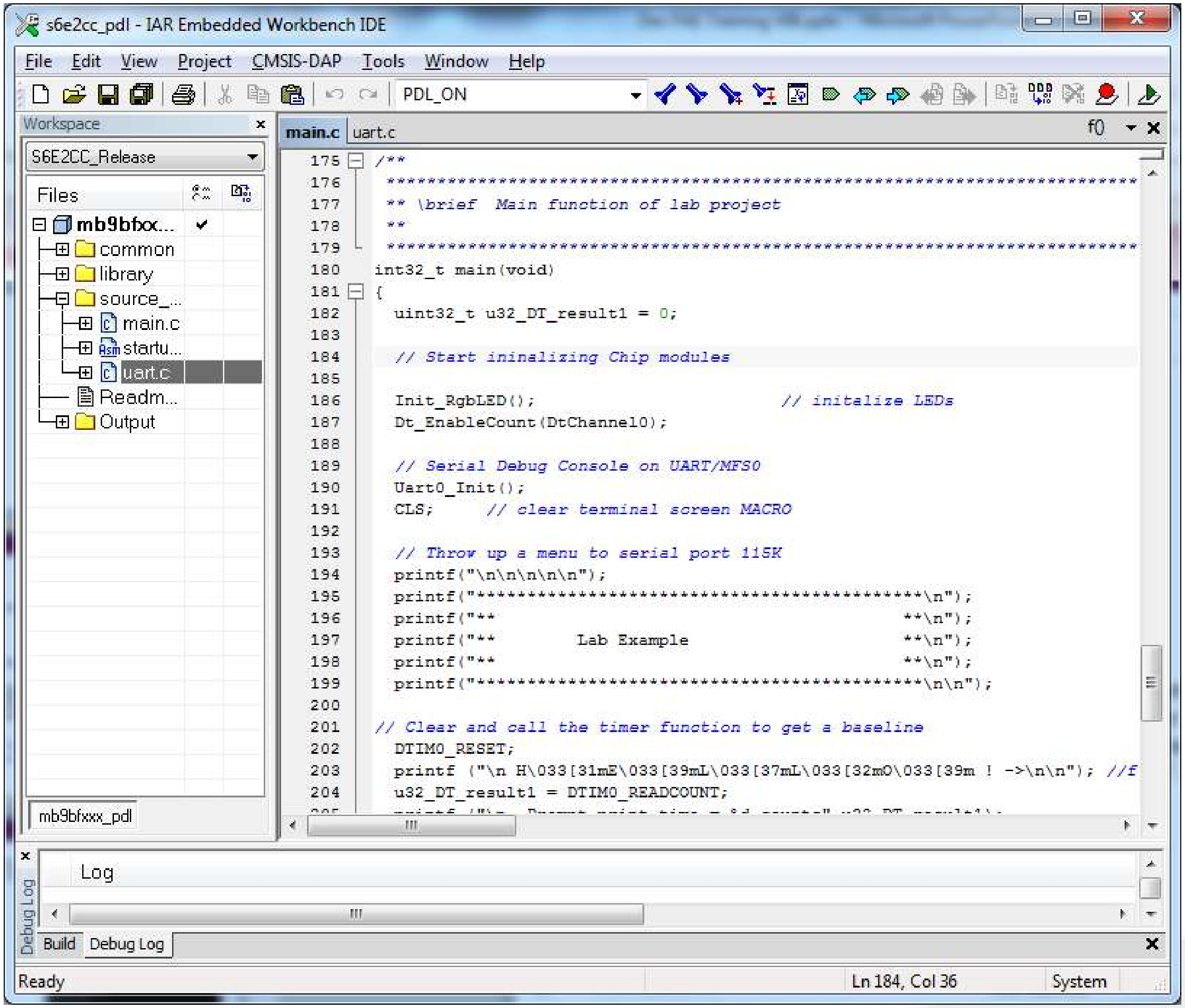
Lab - MCU Template UART -> C printf

The complete clean finial project and workspace is located here:

Launch IAR and open the workspace at:
LABs\uart-printf_v10\uart-printf\uart\template\IAR\SK_s6e2cc_McuUart.eww

Transition to printf

Launch IAR (or close any open workspaces) and open the workspace at:
LABs\uart-printf_v10\uart-printf\printf\template\IAR\s6e2cc_printf.eww



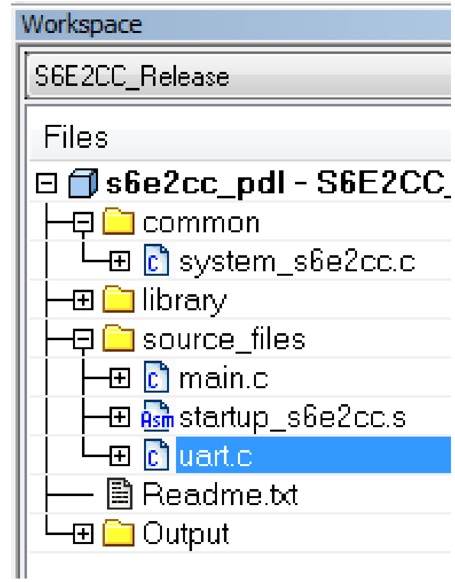
Printf example

So what is different for this new project?

1. Added uart.com to workspace source files.

2. Added several include files
   - **#include "uart.h"**
   - **#include <stdio.h>**
   - **#include <stdlib.h>**

3. Added call to init Serial Debug Console on UART/MFS0
   - **Uart0_Init();**



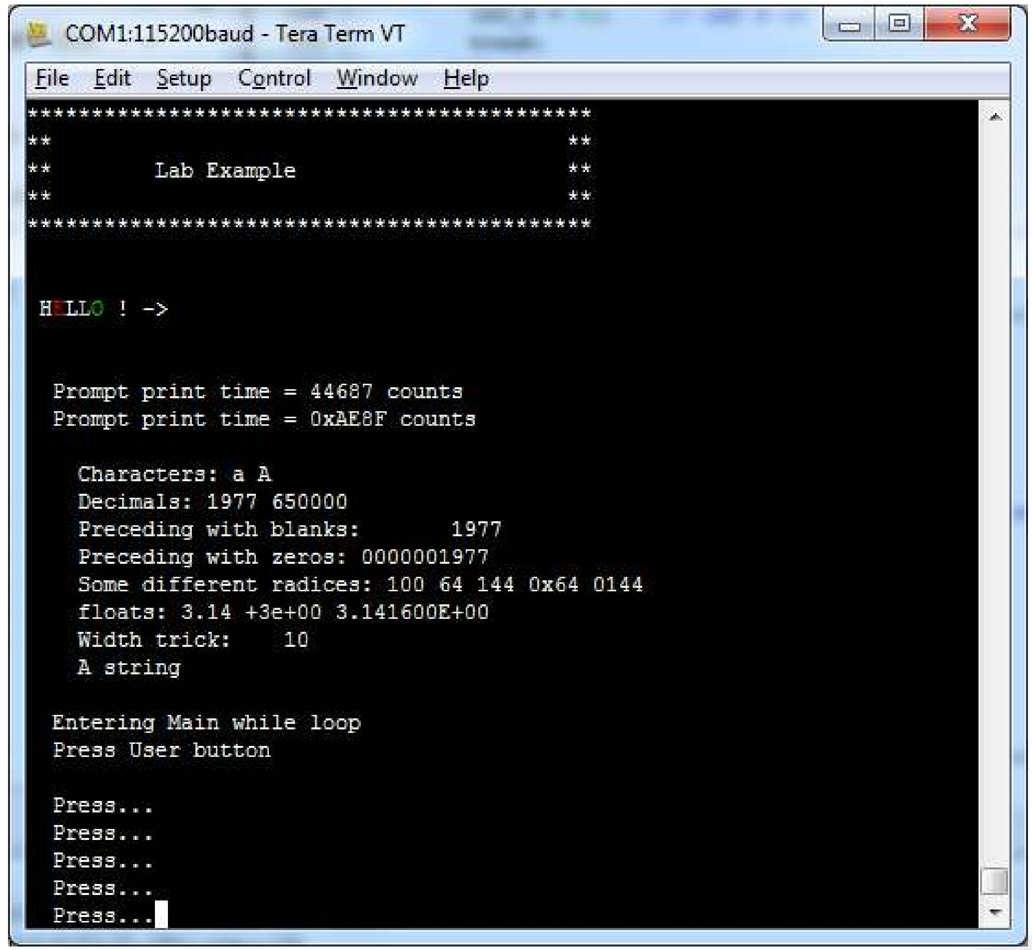
Notice there is a little more than just the printf. To have something to print, we enable the Dual Timer to give us a 100 MHz (half CPU speed) counter that can be used to time events. Just initialize with the Dt_EnableCount function, then the DTIM0_RESET macro clears and starts the counter and the DTIM_READCOUNT macro returns the u32 result.

```
Dt_EnableCount(DtChannel0);

// Clear and call the timer function to get a baseline
DTIM0_RESET;
u32_DT_result1 = DTIM0_READCOUNT;
```

From Project pull-down select rebuild all and then Launch the debugger with the green GO button

Notice the easy, clean, reusable, serial output. Below is the compare of the code and the output on the display.

```
//Some prinf examples
  printf ("    Characters: %c %c \n", 'a', 65);
  printf ("    Decimals: %d %ld\n", 1977, 650000L);
  printf ("    Preceding with blanks: %10d \n", 1977);
  printf ("    Preceding with zeros: %010d \n", 1977);
  printf ("    floats: %4.2f %+.0e %E \n", 3.1416, 3.1416, 3.1416);
  printf ("    Width trick: %*d \n", 5, 10);
  printf ("    %s \n", "A string");
 //now back to the program
```



For details of all the formatting options and possibilities for printf, here is a good link :
http://www.cplusplus.com/reference/cstdio/printf/

Lab - MCU Template UART -> C printf

Results

- Using the normal configuration of the C/C++ runtime library the code size between our MCU UART and Printf was not that significant.
  - 8000 byte ro for printf, 3500 byte for low level – basically printf cost 5K in ROM
  - 1000 byte rw for both (mainly stack).
- It is up to the designer whether code size or ease of portability is the way to implement a serial I/O in a system
- Please understand that the intent with this lab is to start with a simple example that many people can follow. It is not to debate the merits of serial I/O techniques.

Lab - MCU Template UART -> C printf

What just happened?

- We briefly discussed the Multifunction Serial Interface.
- We looked at simple I/O using the MCU template and simple UART.
  - Had a very good intro to debugging session.
- We looked at a more involved implementation using C printf function.
- We learned about some of printf formatting and output capabilities.

SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Dual Flash / Programming via RAM

SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Dual Flash / Programming via RAM

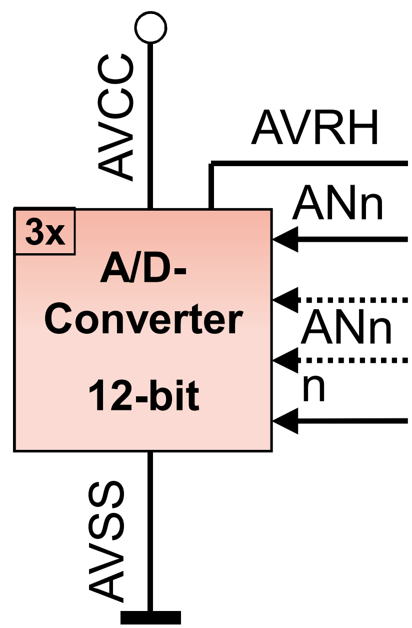
Lab - ADC Example Using the Light Sensor

ADC Example Using the Light Sensor

SK-FM4-176L-S6SE2CC A/D-Converter: Features

A/D-Converter: Fe

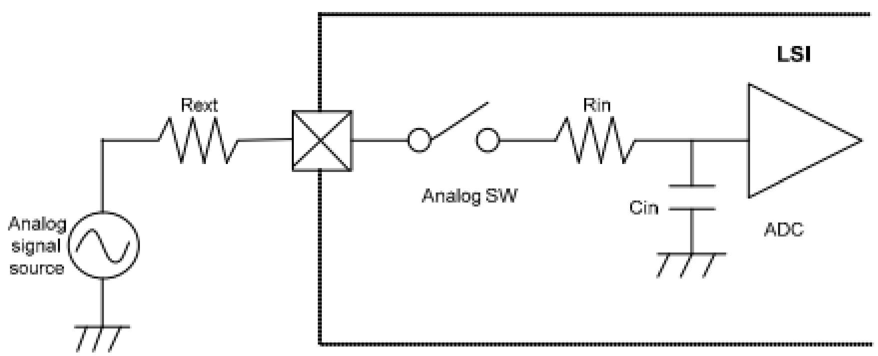
- Up to three A/D converters with 12-bit resolution
  - RC successive approximation
  - Conversion time : 0.5µs @ 5V (see datasheet)
- Up to 24 input channels (depending on package size)
- Two FIFO result buffers
- Three trigger conditions
- Two conversion modes
- Three priority levels
- Comparison function (≥, <)
- Supports DMA transfer
- Range Comparison



ADC Example Using the Light Sensor

SK-FM4-176L-S6SE2CC A/D-Converter : Input channel

- Up to 32 input channels
  - Each channel can be routed to any converter unit
  - Two programmable sampling times selectable
- Programmable conversion time (Sampling + Comparison time)
- After reset ANxy are enabled by default (see register ADE)
  - Pin and code wizard will help you with this
- Analog inputs (**`ADEx = 1`**) cannot be used as digital inputs
  - '0' will be read



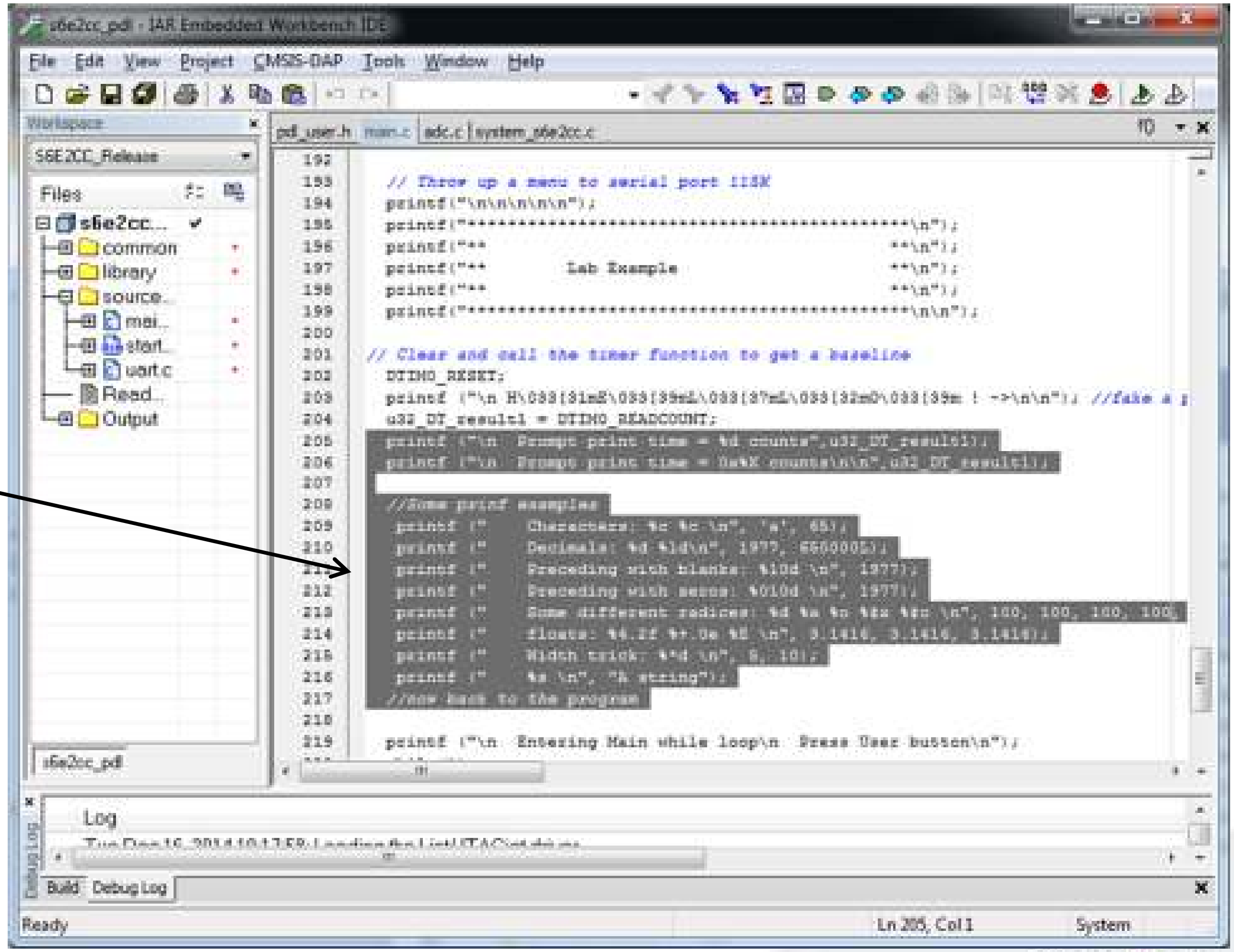
ADC Example Using the Light Sensor

SK-FM4-176L-S6SE2CC A/D-Converter : Running ADC Light

Continuing with the printf project: \Labs\uart-printf\printf\template\IAR\s6e2cc_printf.eww

We are starting from the printf project we just finished. The first thing we will do is remove some of the printf stuff and then we will add in ADC.

Enter the edit window and remove the large section of printf tests.



  CONFIDENTIAL

ADC example

Locate ADC code from examples folder under ADC:
Labs\ADC light\S6E2CC_PDL v0.2\example\adc\adc_scan_polling_sw\main.c
Open it in notepad or other editor (IAR will work too)
Locate the Main_ADC_polling() function and copy it

```
/**
 ******************************************************************************
 ** \brief  ADC0 initialization and single conversion start
 **
 ** \return  uint32_t   ADC result

 ******************************************************************************/
uint32_t Main_adc_polling(void)
{
  stc_adcn_t*      pstcAdc = NULL;
  stc_adc_config_t stcConfig;
  uint32_t         u32Result;
...
}
```

Paste it in he open IAR project in main.c right above the main function in the project.

```
/**
 ******************************************************************************
 ** \brief  Main function of lab project
 **

 ******************************************************************************/
int32_t main(void)
{
  uint32_t u32_DT_result1 = 0;
```

ADC Example Using the Light Sensor

Rename the Main_ADC_polling to Init_ADC. Cut the local definition of the ADC structure and place it in the Global variable definitions at the top of main about line 100. This will make the ADC structure accessible across functions.

```
/**
 ******************************************************************************
 ** \brief  ADC0 initialization and single conversion start
 **
 ** \return  uint32_t   ADC result

 ******************************************************************************/
uint32_t Init_adc(void)
{
  stc_adcn_t*       pstcAdc = NULL;
  stc_adc_config_t stcConfig;
  uint32_t          u32Result;
...

  return u32Result;
}
```

Add the Init_adc(); and a variable to hold the result into the main function.

```
int32_t main(void)
{
 uint32_t u32_DT_result1 = 0;           // Dual Timer variable
 volatile uint32_t u32AdcResult = 0;    // global ADC result variable

  // Start initializing Chip modules
  Init_ADC();                            // initialize ADC
  Init_RgbLED();                         // initialize LEDs
  Dt_EnableCount(DtChannel0);            // initialize Dual timer.
```

ADC Example Using the Light Sensor

From the bottom of Init_ADC() pull the four lines below out of the function (cut)

```
if (Ok == Adc_Init(&ADC0, &stcConfig))        // Init ADC0
 {
   pstcAdc = (stc_adcn_t*) &ADC0;

   Adc_EnableWaitReady(pstcAdc);                // Enable ADC0 and wait for ready

   Adc_ScanSwTrigger(pstcAdc);                  // Trigger ADC0
   while (OperationInProgress == Adc_ScanStatus(pstcAdc))      // Poll for conversion done
   {}
   u32Result = Adc_ReadScanFifo(pstcAdc);
 }
```

Add these lines to the main() function under the switch detection

```
while(1)
 {
   if (0u == bFM4_GPIO_PDIR2_P0)  //user button pressed
   {
   Adc_ScanSwTrigger(pstcAdc);                  // Trigger ADC0
   while (OperationInProgress == Adc_ScanStatus(pstcAdc))      // Poll for conversion done
   {}
   u32Result = Adc_ReadScanFifo(pstcAdc);
   while(0u == bFM4_GPIO_PDIR2_P0) // Wait for SW2 release
     {}
   printf ("\n  Press...");

   u8LedState++;

   if (u8LedState > 7u)
     {
       u8LedState = 0u;
     }
   Update_LED();
   }
 }
```


CONFIDENTIAL

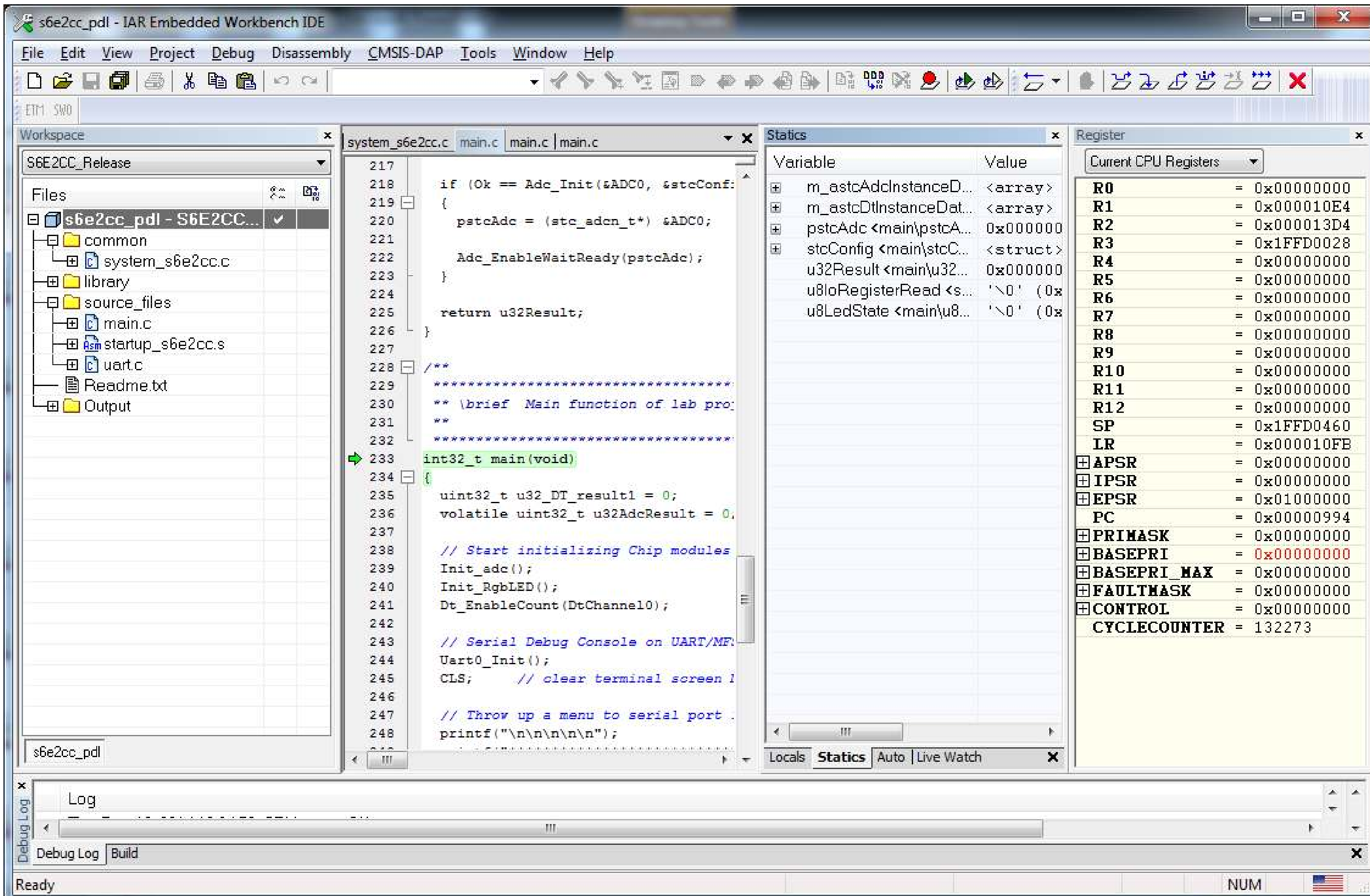
ADC Example Using the Light Sensor

Modify the nearby printf to output the ADC result.

```
while(1)
 {
   if (0u == bFM4_GPIO_PDIR2_P0)  //user button pressed
   {
   Adc_ScanSwTrigger(pstcAdc);                   // Trigger ADC0
   while (OperationInProgress == Adc_ScanStatus(pstcAdc))      // Poll for conversion done
   {}
   u32Result = Adc_ReadScanFifo(pstcAdc);
   while(0u == bFM4_GPIO_PDIR2_P0) // Wait for SW2 release
     {}
//   printf ("\n  Press...");
    printf ("\n  Sensor = %d counts",u32_ADC_result1>>16);  //print ADC value
   u8LedState++;

   if (u8LedState > 7u)
     {
       u8LedState = 0u;
     }
   Update_LED();
   }
 }
```

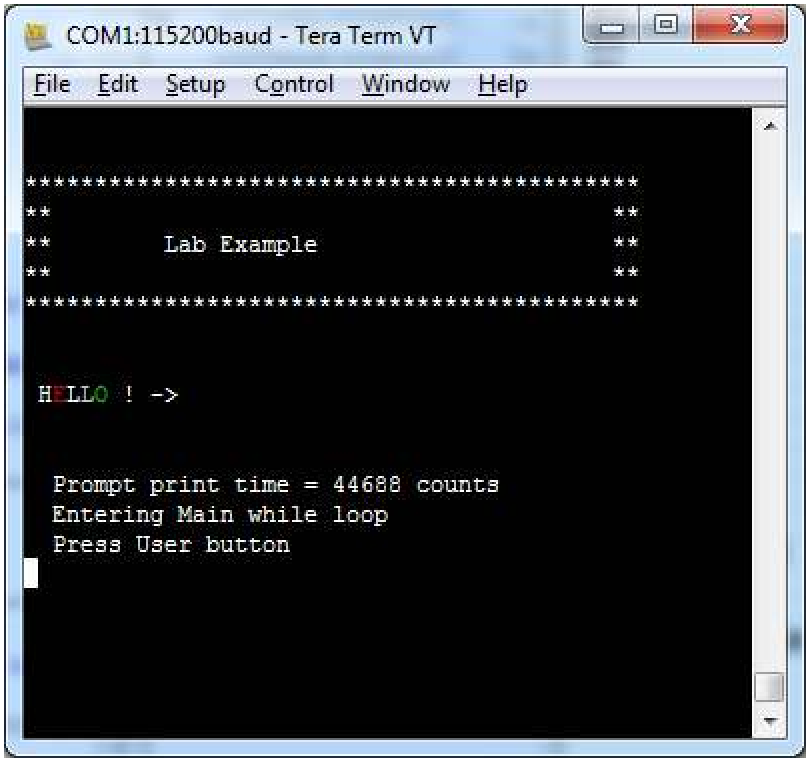
ADC Example Using the Light Sensor

Select project pull down from the title menu and then click rebuild all.
If build is successful launch the debugger with the  button in title bar.

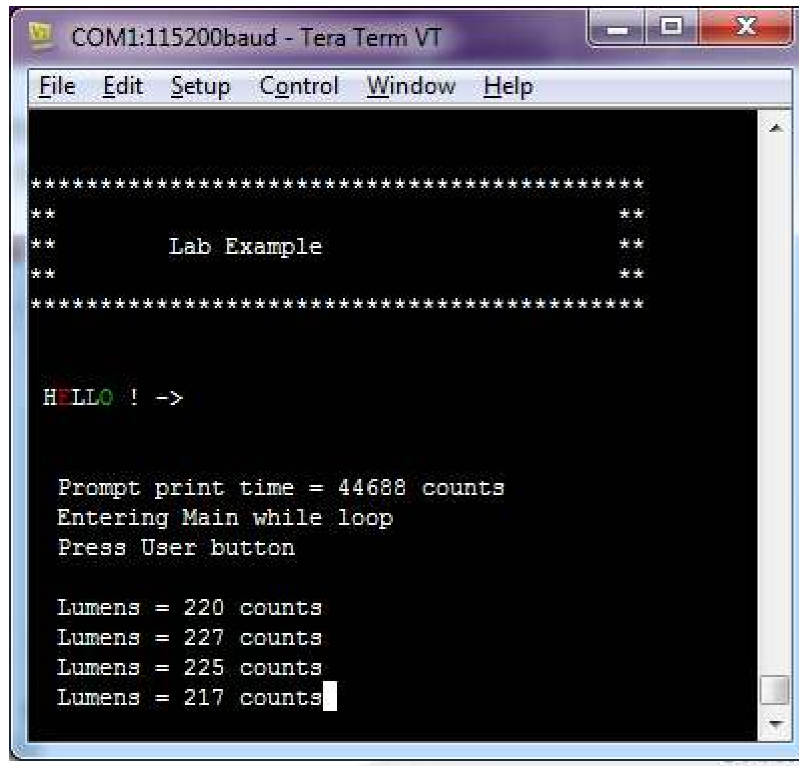


ADC example – Looks much better now.

Open your favorite Terminal emulator and start the program running with the Go button on the title bar.



Pressing the user button will trigger a new ADC conversion and display the results



ADC example

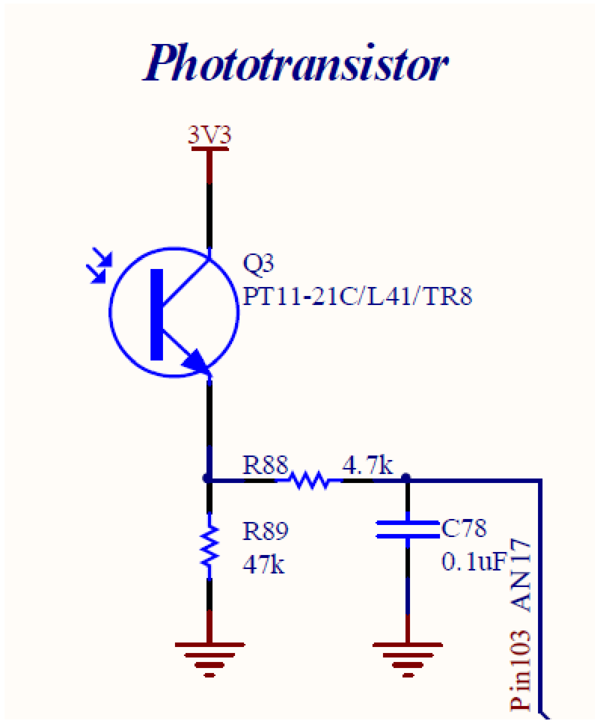
Does something seem funny to you?

The system does not seem to be very responsive to changes in light intensity. It does vary some but it just does not seem like a good reading.

Can anyone Identify an error in the captured sensor of the schematic?

I hope not. There is no error. What gives?

Notice that Q3 is connected to AN17. Hmmm



ADC example

Let's go back and look more closely at the ADC structure we just integrated into our code. Notice anything out of order?

```
/**
 ******************************************************************************
 ** \brief  ADC0 initialization and single conversion start
 **
 ** \return  uint32_t   ADC result
 ******************************************************************************/
uint32_t Init_adc(void)
{

  PDL_ZERO_STRUCT(stcConfig);                // Clear local configuration to zero.

  stcConfig.u32CannelSelect.AD_CH_0 = 1u;
  stcConfig.bLsbAlignment = TRUE;
  stcConfig.enScanMode = ScanSingleConversion;
  stcConfig.u32SamplingTimeSelect.AD_CH_0 = 1u;
  stcConfig.enSamplingTimeN0 = Value32;
  stcConfig.u8SamplingTime0 = 30u;
  stcConfig.enSamplingTimeN1 = Value32;
  stcConfig.u8SamplingTime1 = 30u;
  stcConfig.u8SamplingMultiplier = 4u;
  stcConfig.u8EnableTime = 10u;
  stcConfig.bScanTimerStartEnable = FALSE;
  stcConfig.u8ScanFifoDepth = 0u;
```

ADC example

Notice anything out of order? Other than the spelling of Channel the converter is setup to mux in channel 0, we are on ch17. Let's try changing the channel.

```
/**
 ******************************************************************************
 ** \brief  ADC0 initialization and single conversion start
 **
 ** \return  uint32_t   ADC result
 ******************************************************************************/
uint32_t Init_adc(void)
{

  PDL_ZERO_STRUCT(stcConfig);                // Clear local configuration to zero.

  stcConfig.u32CannelSelect.AD_CH_0 = 1u;
  stcConfig.bLsbAlignment = TRUE;
  stcConfig.enScanMode = ScanSingleConversion;
  stcConfig.u32SamplingTimeSelect.AD_CH_0 = 1u;
  stcConfig.enSamplingTimeN0 = Value32;
  stcConfig.u8SamplingTime0 = 30u;
  stcConfig.enSamplingTimeN1 = Value32;
  stcConfig.u8SamplingTime1 = 30u;
  stcConfig.u8SamplingMultiplier = 4u;
  stcConfig.u8EnableTime = 10u;
  stcConfig.bScanTimerStartEnable = FALSE;
  stcConfig.u8ScanFifoDepth = 0u;
```

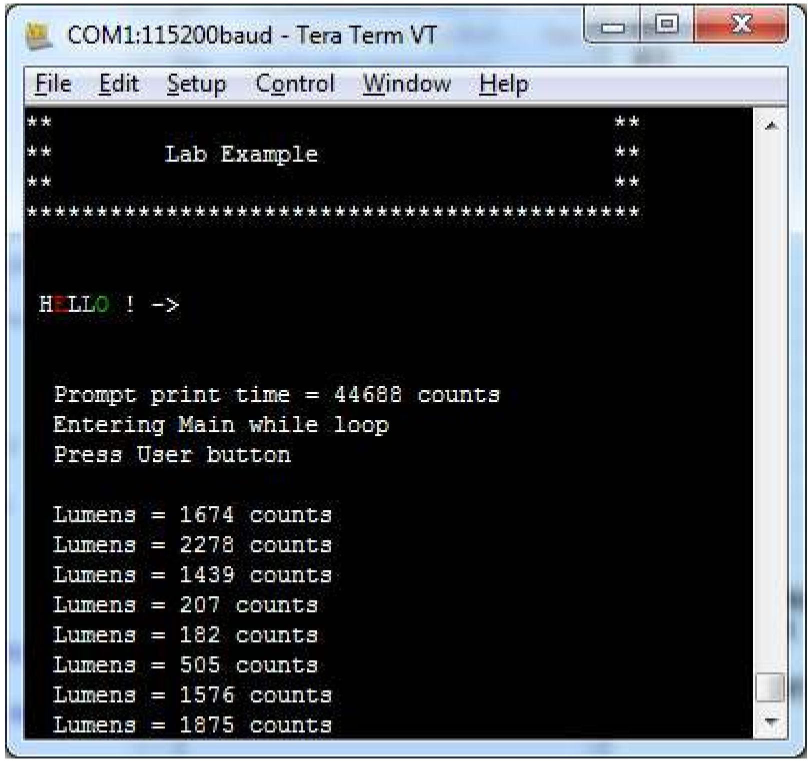
ADC example

Simple change for channel selection

```
/**
 ******************************************************************************
 ** \brief  ADC0 initialization and single conversion start
 **
 ** \return  uint32_t   ADC result
 ******************************************************************************/
uint32_t Init_adc(void)
{

  PDL_ZERO_STRUCT(stcConfig);                 // Clear local configuration to zero.

  stcConfig.u32CannelSelect.AD_CH_17 = 1u;
  stcConfig.bLsbAlignment = TRUE;
  stcConfig.enScanMode = ScanSingleConversion;
  stcConfig.u32SamplingTimeSelect.AD_CH_0 = 1u;
  stcConfig.enSamplingTimeN0 = Value32;
  stcConfig.u8SamplingTime0 = 30u;
  stcConfig.enSamplingTimeN1 = Value32;
  stcConfig.u8SamplingTime1 = 30u;
  stcConfig.u8SamplingMultiplier = 4u;
  stcConfig.u8EnableTime = 10u;
  stcConfig.bScanTimerStartEnable = FALSE;
  stcConfig.u8ScanFifoDepth = 0u;
```

ADC example

Ahhh much better.



The full completed solution is available at:
Labs\ADC light\template\IAR\SK-s6e2cc_ADC_light.eww

ADC example

Conclusion

- We briefly discussed the Analog-to-Digital Interface.
- We used one of the PDL examples to modify the printf example.
- We used the ADC to read the light sensor on the starter kit.

SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Quad SPI flash interface
- Lab - Dual Flash / Programming via RAM

SK-FM4-176L-S6SE2CC List of Labs

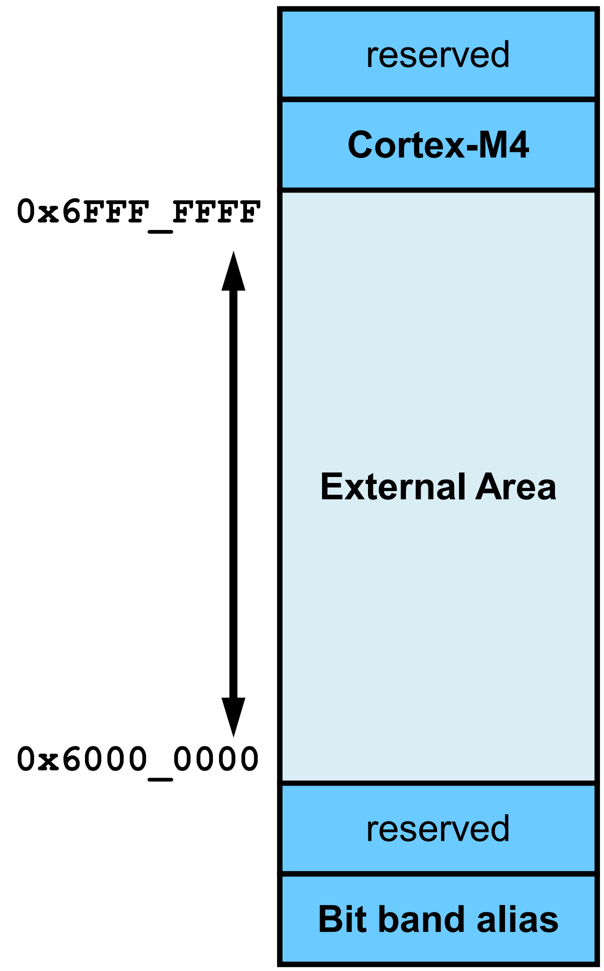
- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Quad SPI flash interface
- Lab - Dual Flash / Programming via RAM

External Bus Interface (EBI) Lecture

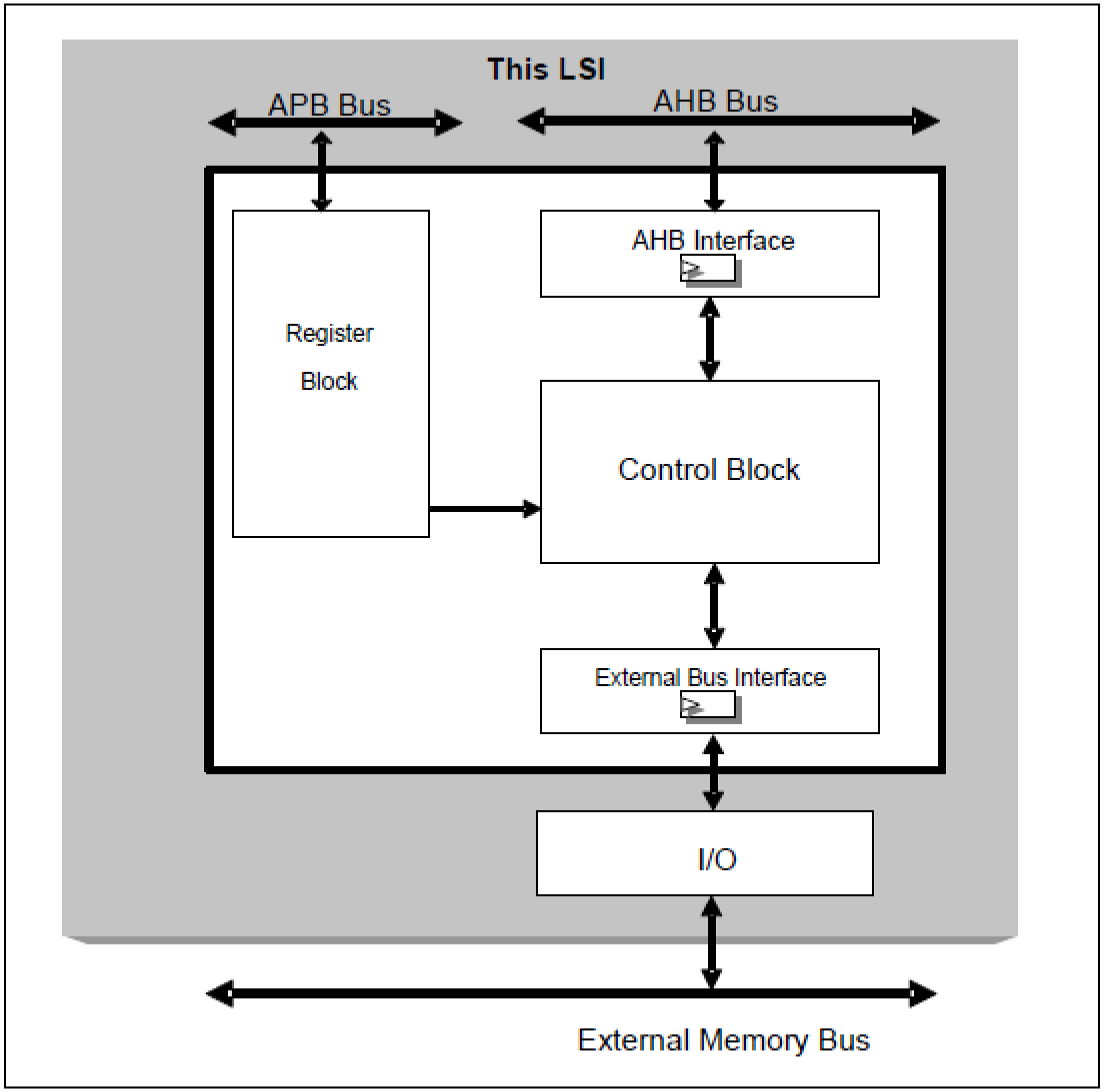
External Bus Interface Module.
Reviewed the various memory types
Enable and test the external asynchronous interface

External Bus Interface

- Non-multiplexed and multiplexed bus
  - Up to 256 MB address space
  - Little endian
- 8-bit/16-bit data width
- 8 chip select areas
  - Separate configuration for each area
- Supports NAND flash memory access
- Supports NOR flash/SRAM memory page access
- Supports SDRAM memory access
  - Adjustable refresh and refresh time
  - Power-on and -off sequence

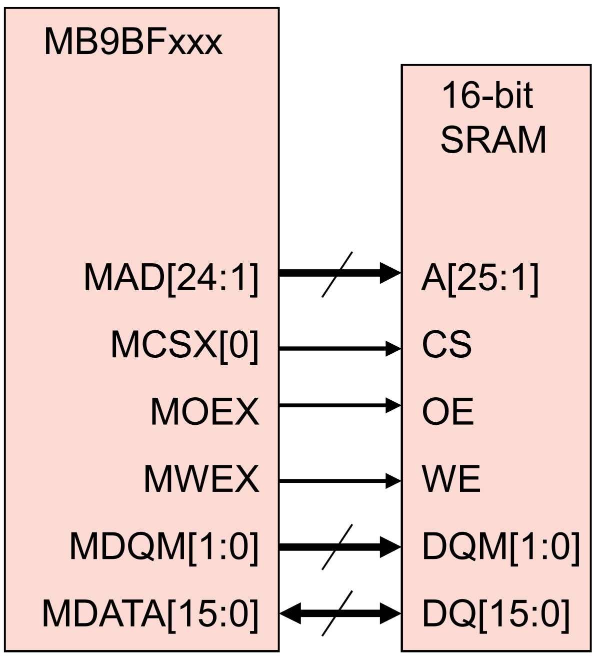


External bus interface block diagram

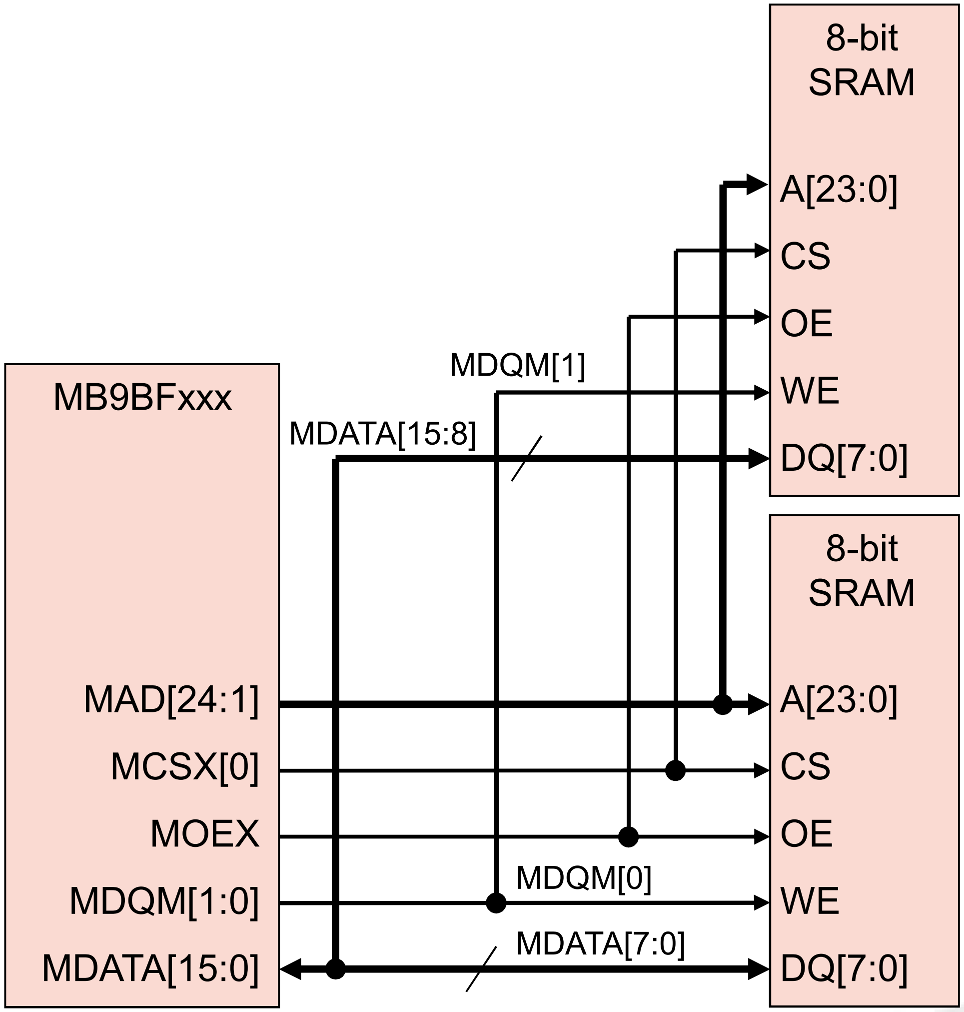


External Bus Interface - Asynchronous Memory access

- Connection examples SRAM
- Newly introduced PSRAM
- Also Paged access for NOR
- Why is A0 not used?



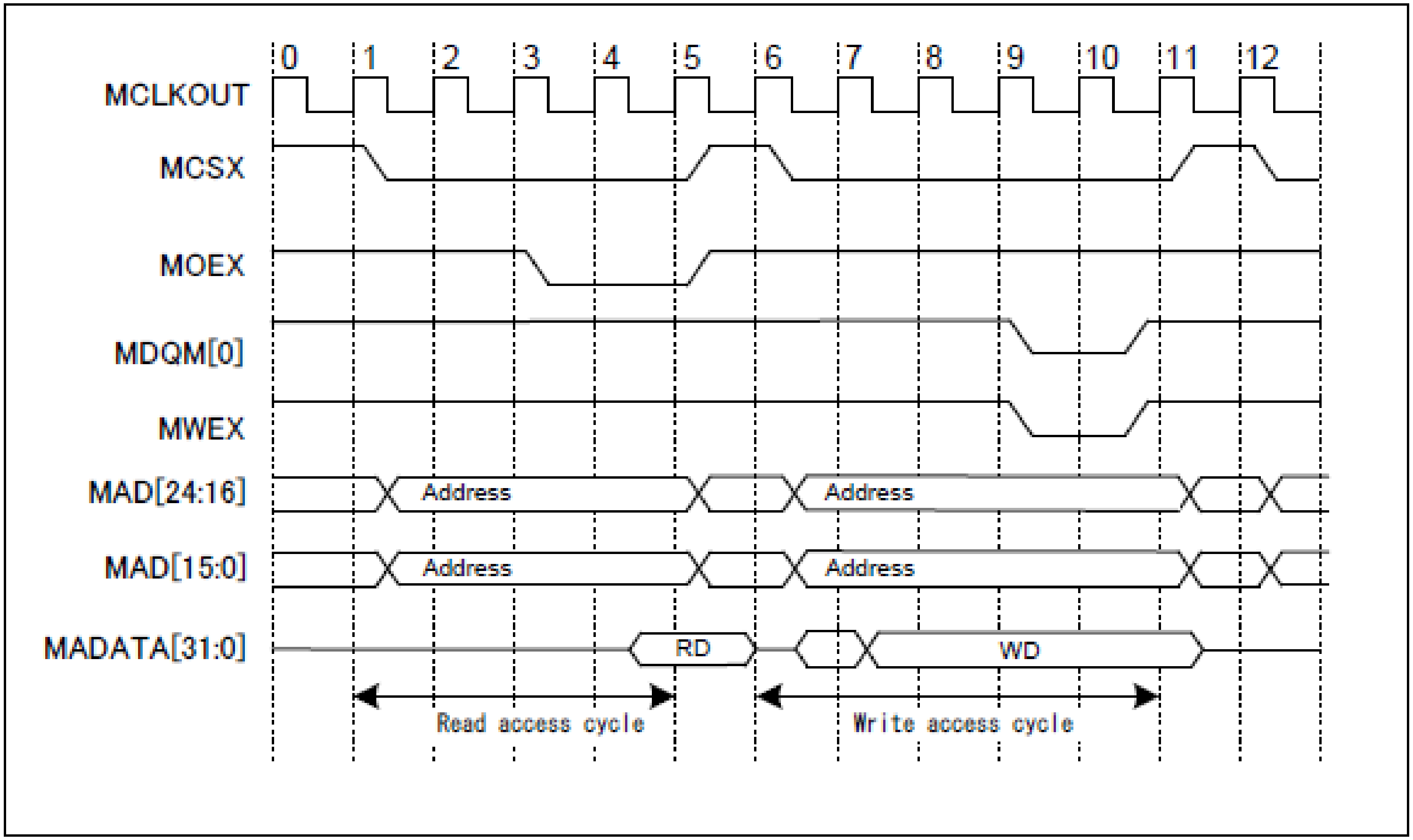
16-bit SRAM connection



8-bit SRAM x2 connection

Basic Asynchronous Memory access

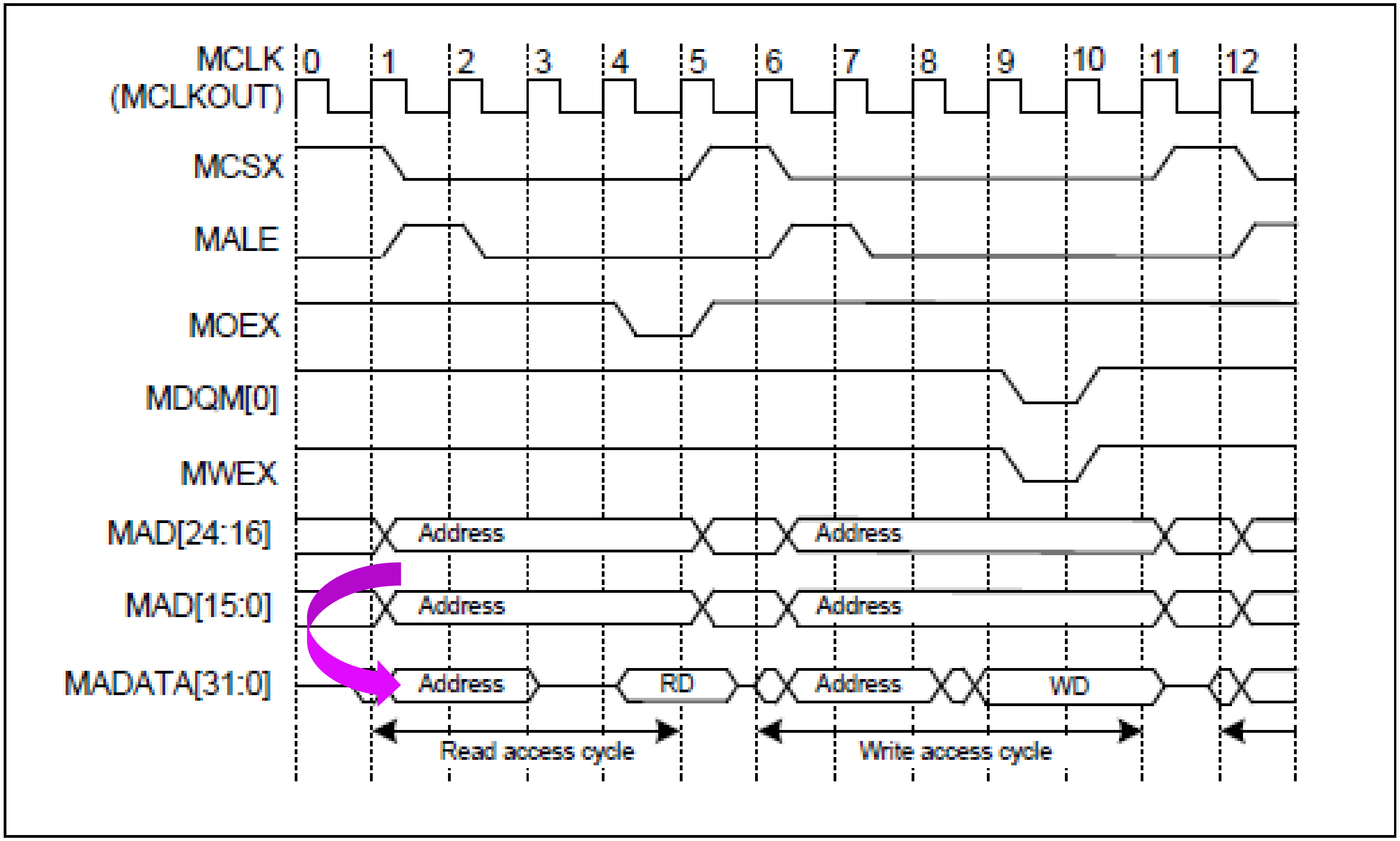
NOR Flash (non-paged), SRAM, PSRAM



  CONFIDENTIAL

External Bus Interface - Asynchronous Memory access

Multiplexed accesses – Saves on number of pins required possibly at the cost of speed.

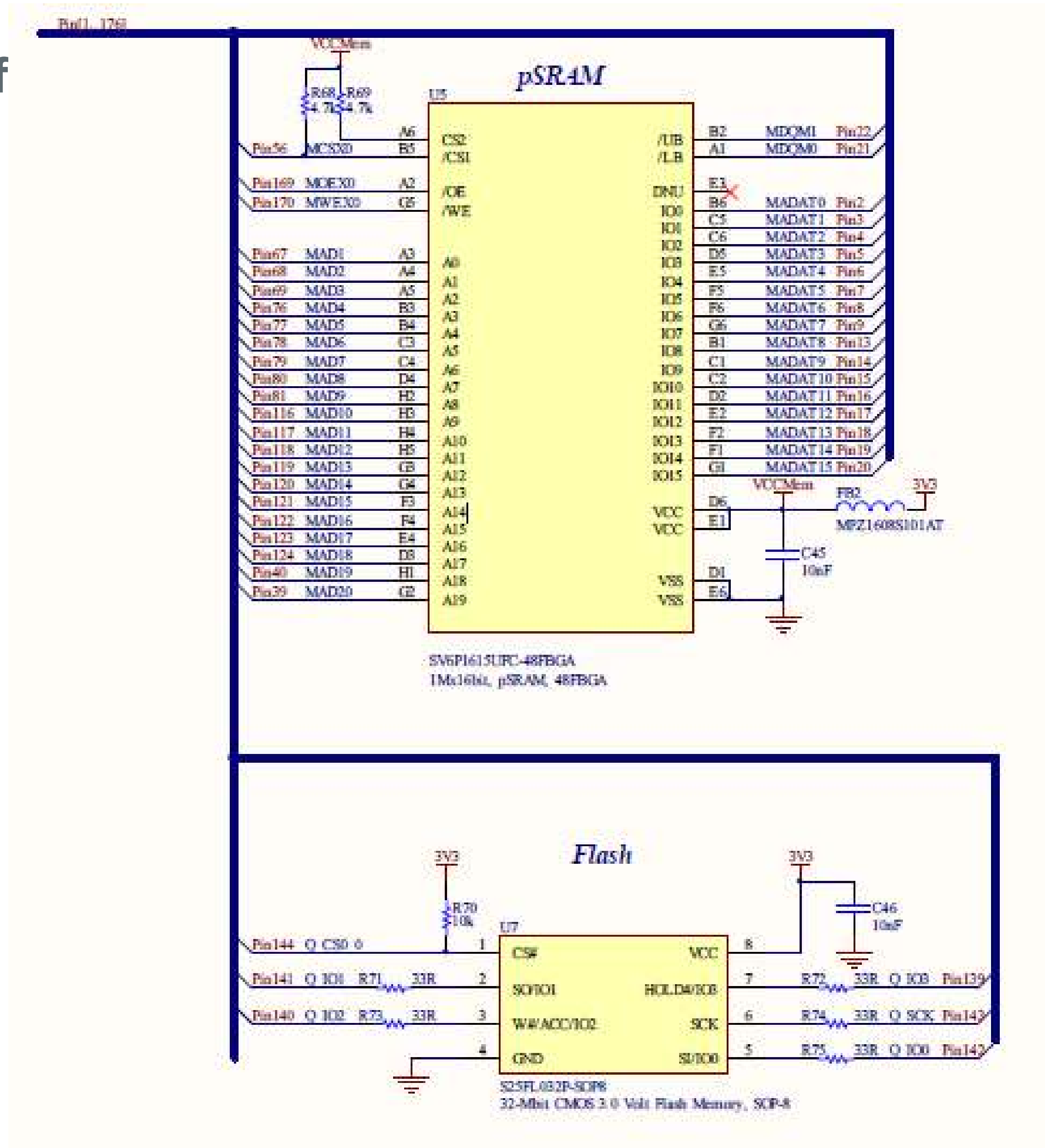


CONFIDENTIAL

External Bus Interface -

Sk-176-S6E2CC specif

- PSRAM Asyc 70ns

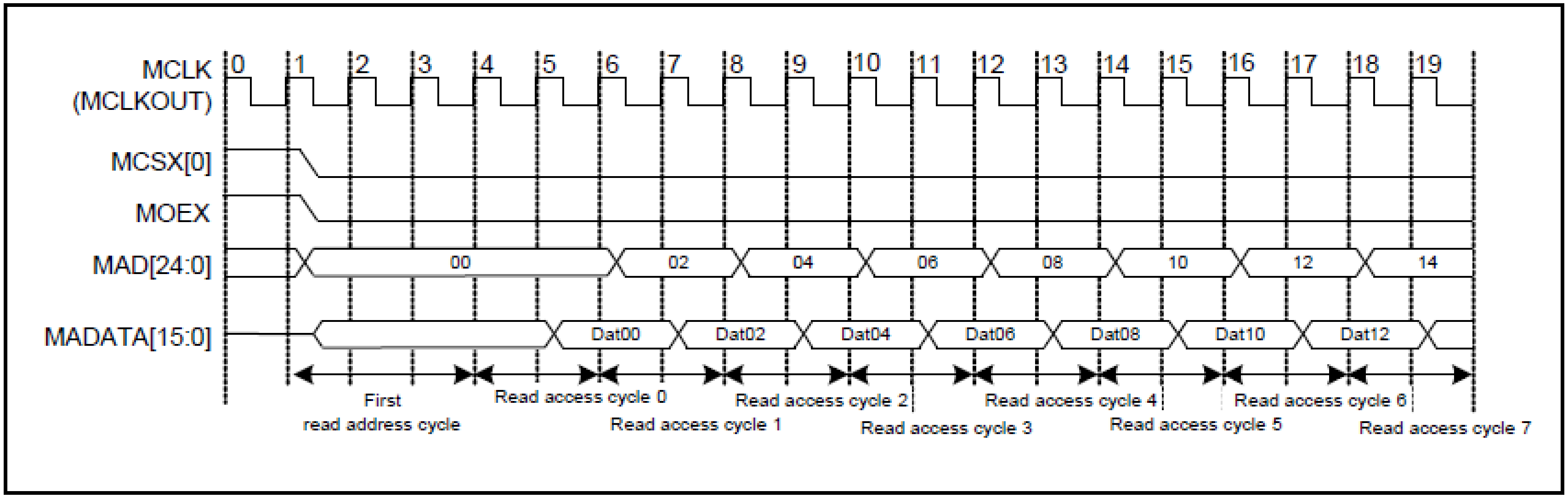


- 32Mbit HS QSPI

Works in real life too.

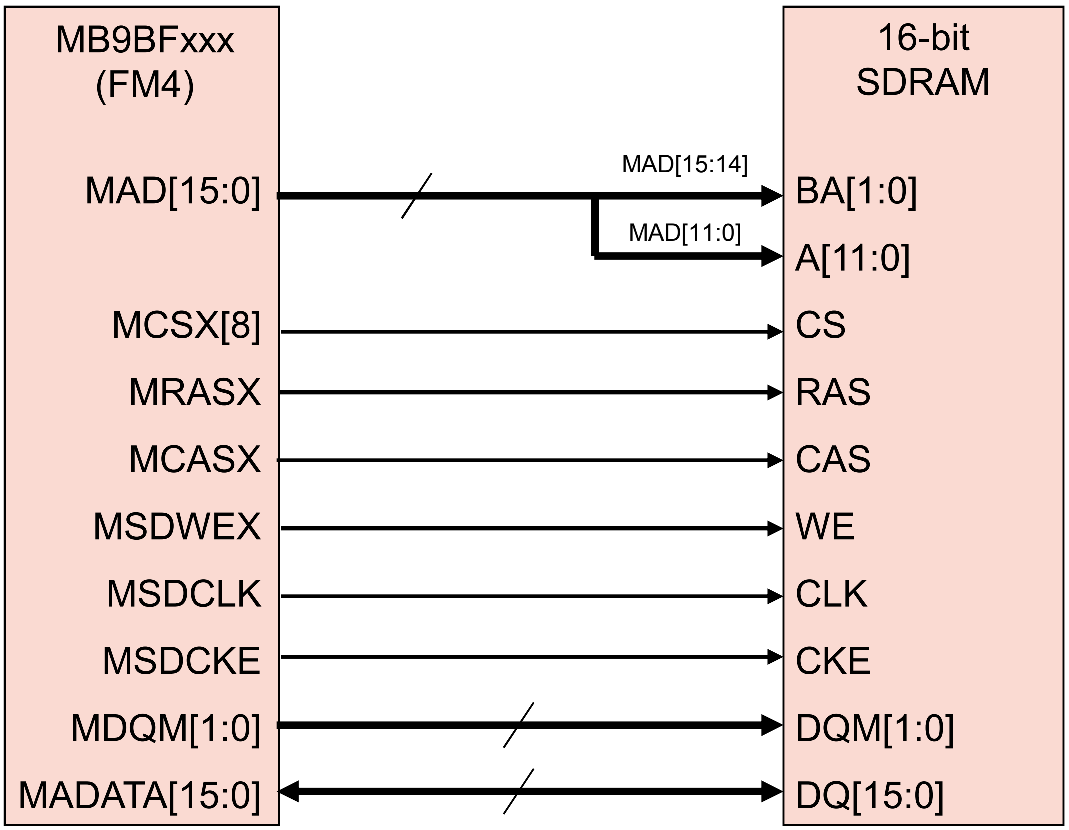
External Bus Interface - Asynchronous Memory access

NOR flash page mode.



External Bus Interface SDRAM

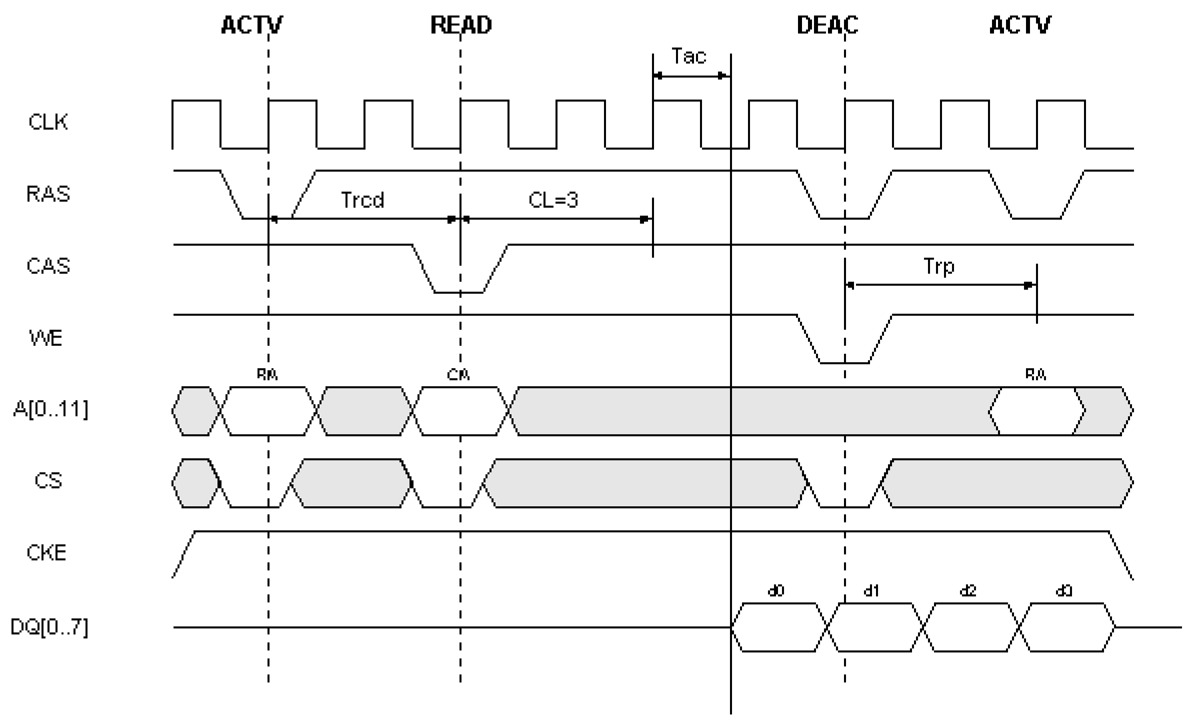
- Connection examples SDRAM (FM4)
  - MCSX[8] Pin is fixed for SDRAM chip enable usage



16-bit SDRAM connection

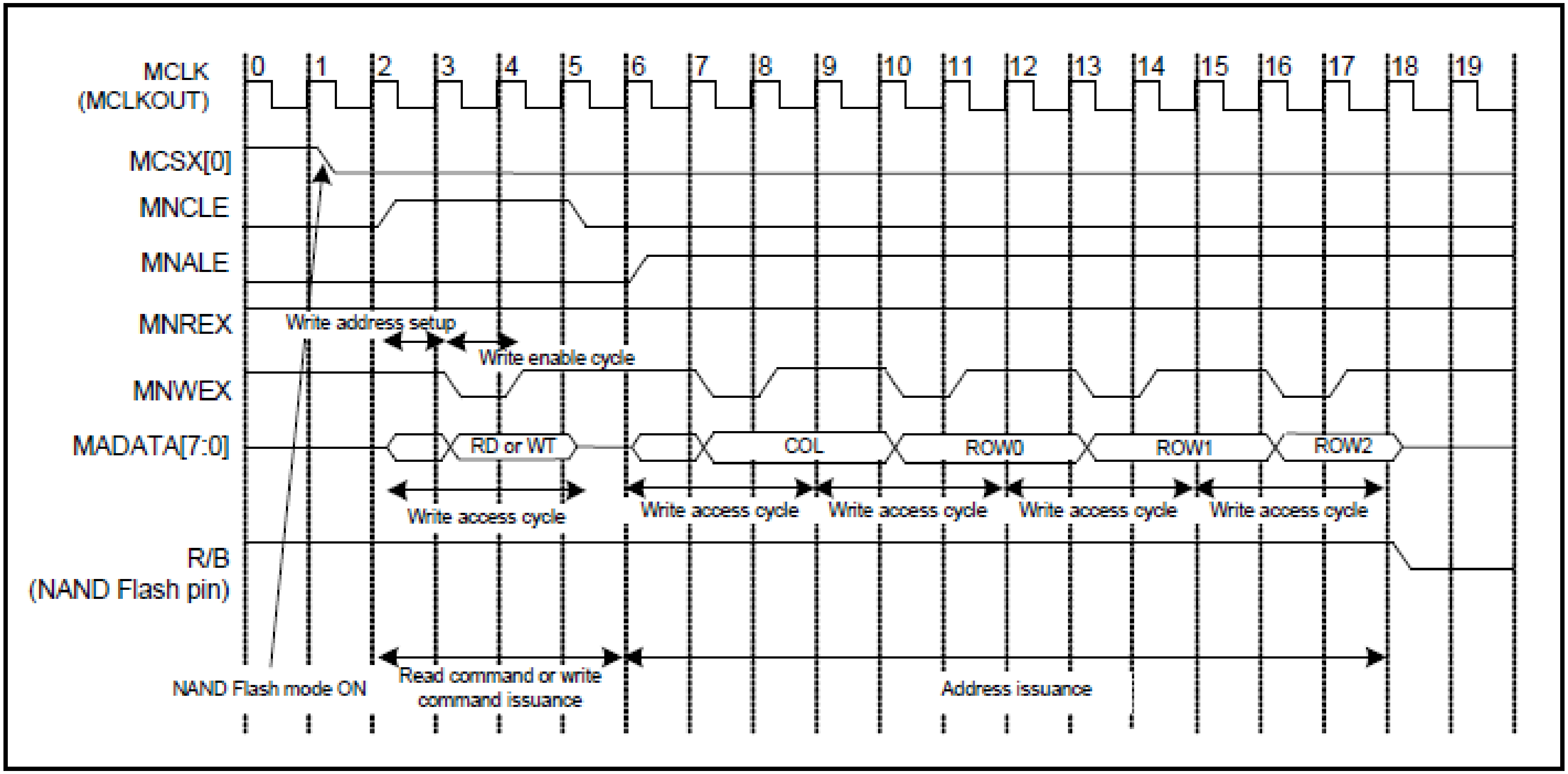
External Bus Interface - SDRAM timing

Link to more info on SDRAM



SDRAM Burst Access

Commanded Access Memory - NAND

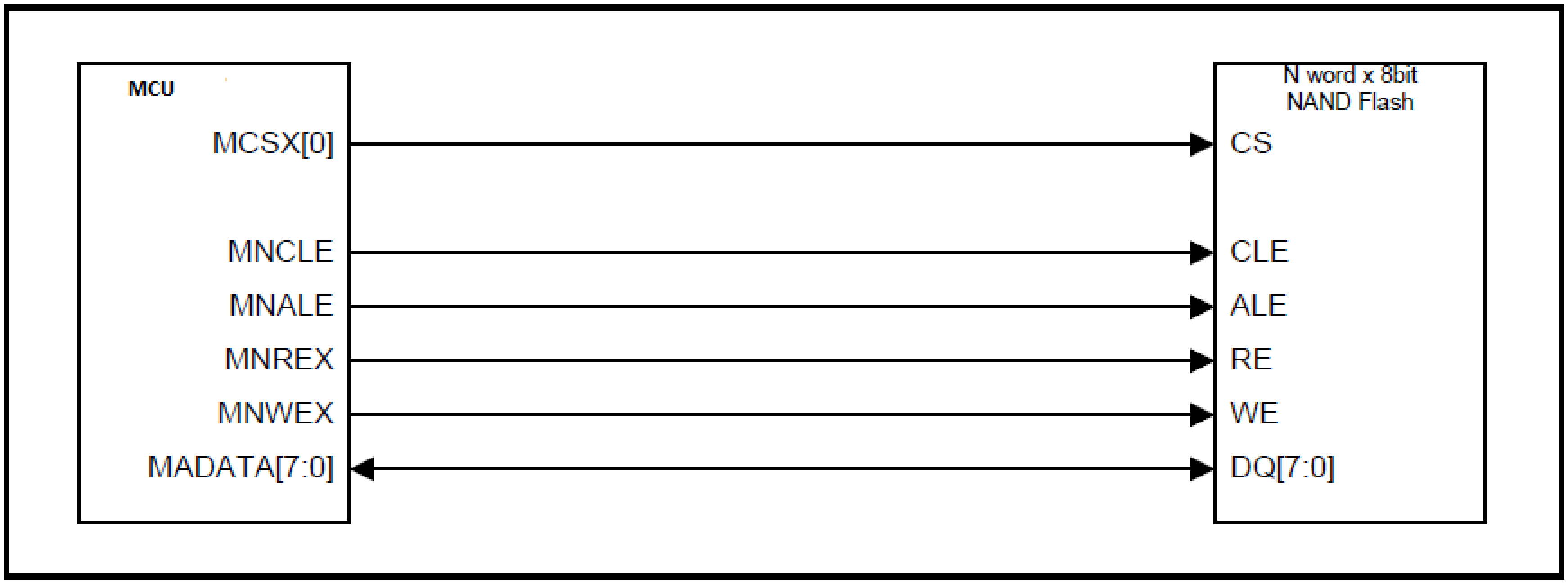


Very clean interface

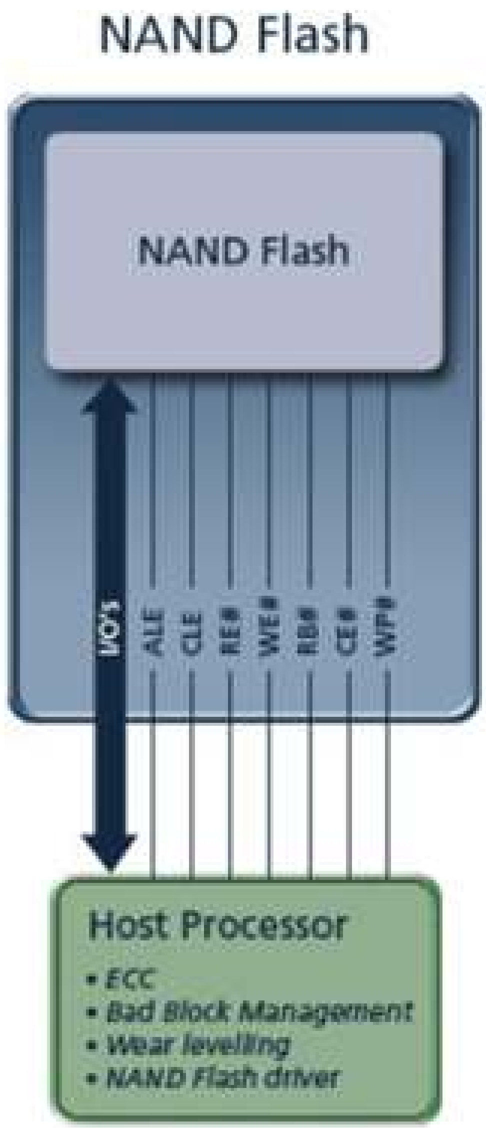
N-word x 8-bit NAND

Example of connecting an N-word x 8-bit NAND Flash memory.

Example of 8-bit NAND Flash memory connection



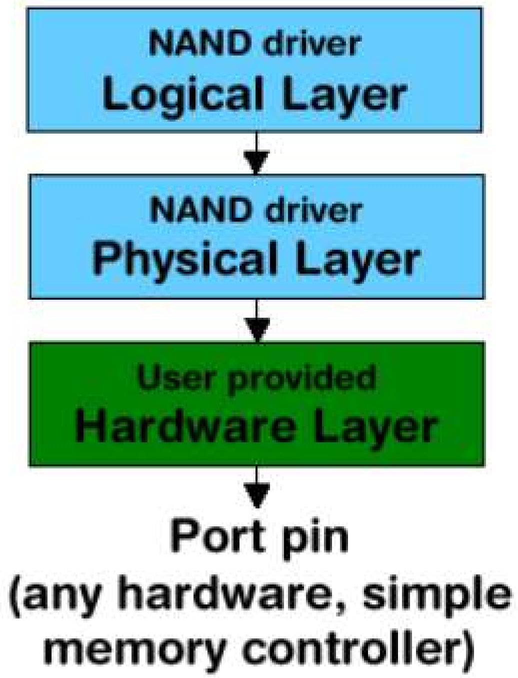
External Bus Interface - NAND FLASH



NAND flash memory requires a controller to aid with array management, ECC, wear leveling, and other functions. But where that controller resides and works can and should vary, based on a specific design's needs.

The EBI interface does not provide NAND flash controller functionality in hardware.

Example commands



The NAND Flash memory is controlled using a set of commands; that set varies from memory to memory. According to ONFI Standard (5) the below list is a basic mandatory command set with their respective command codes (first/second byte).

- Read, 00h/30h
- Change Read Column, 05h/E0h
- Block Erase, 60h/D0h
- Read Status, 70h
- Page Program, 80h/10h
- Change Write Column, 85h
- Read ID, 90h
- Read Parameter Page, ECh
- Reset, FFh

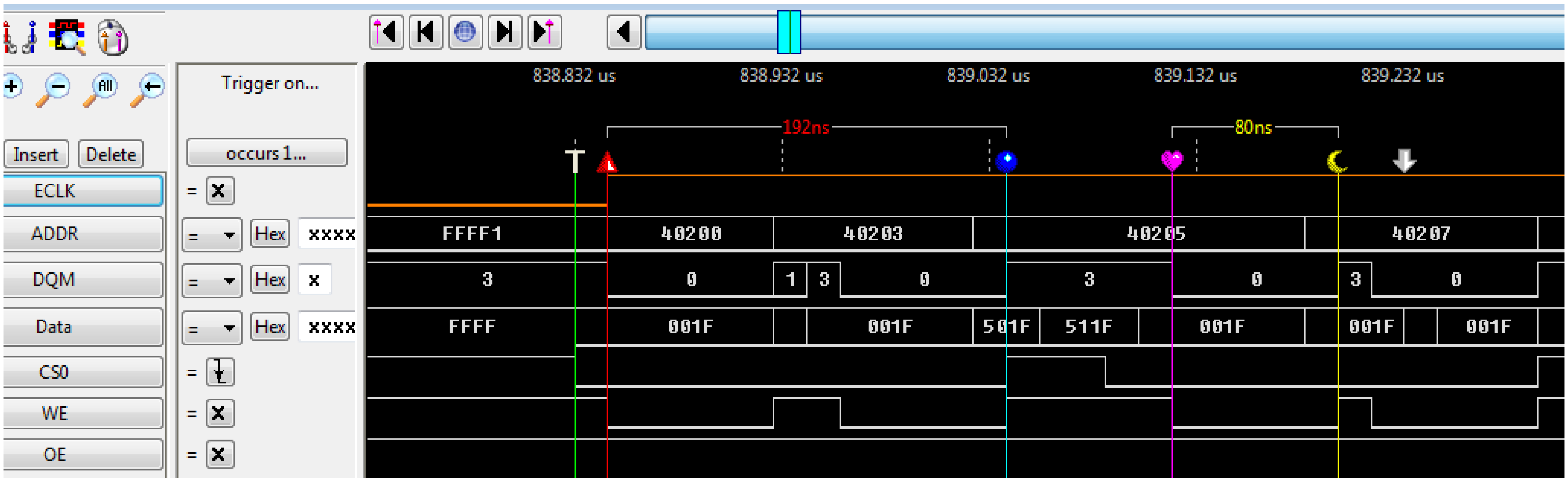
External Bus Interface - MCU template

```
/*****************************************************************************
*  DESCRIPTION:    initialize external bus I/F
*  PARAMETERS:     None
*  RETURNS:        None
*****************************************************************************/
void Ebi_Init(void)
{
    // Setup external Bus with 16Bit PSRAM
    // The bus runs @ 50Mhz (about 3cycle, so the cycle - time about 63ns

// Set Clocking and Access Mode.
     FM4_EXBUS->DCLKR = 0x00000013; // turn on bus clock output /4 200/4= 50MHz
     FM4_EXBUS->AMODE = 0x00000000; //Disables the preceding read and continuous write request
     // No ADC pins in the way right now..
     FM4_GPIO->ADE  =  0x00000000; // no ADC pins
     // Set up the pin function registers.
     FM4_GPIO->PFRF |= 0x00000080; //MADCLK
     FM4_GPIO->PFRA |= 0x0000FFFF; //MADATA00-15
     FM4_GPIO->PFR7 |= 0x000047FE; //MAD01-09,MCSX7,cs0
     FM4_GPIO->PFR6 |= 0x0000000C; //MOEX,MWEX
     FM4_GPIO->PFR3 |= 0x00006000; //MAD19-20
     FM4_GPIO->PFR2 |= 0x000007F0; //MAD12-18
     FM4_GPIO->PFR1 |= 0x0000C000; //MAD10-11
     FM4_GPIO->PFR0 |= 0x00000700; //MDQM0-1,Bus Clk
 //Mode - Setting for /CS0
    //Bit 31..14  reserved
    //Bit 13  MOEXEUP 0     MOEX width is set with RACC-RADC
    //Bit 12  MPXCSOF 0     Asserts MCSX in ALC cycle period
    //Bit 11  MPXDOFF 1     Do not output the address to the data lines
    //Bit 10  reserved
    //Bit 9   ALEINV  0     ALE Inverter mode is turned off
    //Bit 8   MPXMODE 0     MPLEX MODE is turned off
    //Bit 7  SHRTDOUT 1     Short Data Out mode is turned on
    //Bit 6     RDY   0     RDY Mode On is turned off
    //Bit 5     PAGE  0     NOR - page mode is turned off
    //Bit 4     NAND  0     NAND - Flash mode is turned off
    //Bit 3     WEOFF 0     Enable MWEX - Signal
    //Bit 2     RBMON 1     Disable read byte mask
    //Bit 1-0    WDTH 0 1     16 bit Data - bus
     FM4_EXBUS->MODE0 = 0x00000885;

    //Timing Register for /CS0
    //Bit 31..28  WIDLC 0   Write Idle Cycle +1
    //Bit 27..24  WWEC  3   Write Enable Cycle +1
    //Bit 23..20  WADC  0   Write Address Setup cycle +1
    //Bit 19..16  WACC  4   Write Access Cycle !=0
    //Bit 15..12  RIDLC 0   Read Idle Cycle +1
    //Bit 11..08  FRADC 0   First Read Address Cycle +1
    //Bit 07..04  RADC  0   Read Address Setup cycle
    //Bit 03..00  RACC  3   Read Access Cycle +1
     FM4_EXBUS->TIM0 =  0x03040003;   //EXBUS->TIMing register;
     FM4_EXBUS->AREA0 = 0x003F0000;    //Start CS0 0x60000000 length 1MB

     // Enable Address external Bus pins
     FM4_GPIO->EPFR10 = 0x0FFF803F;  //MAD08-20, nMAD00, nCS1-7, nNAND, MOEX, nMDQM0/1, MWEX, MCLKOUT, WDH/RDH, WDL/RDL
     FM4_GPIO->EPFR11 = 0x01FFFFFE;  //No re-locate, MADATA00-15, MAD01-07, MCSX0, nMALE.
     FM4_GPIO->EPFR20 = 0x00000000;  //No MADATA16-MADATA31, No
}
```

Reviewing the reference manual you will see there are not that many registers shown here in RED. However there is just eight of each 0-7 for each of the definable memory areas.

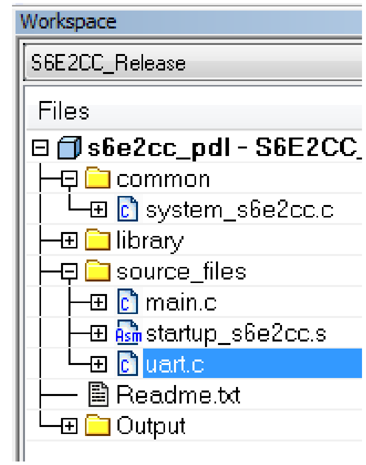
External Bus Interface – Always validate your setup

At some point in your external bus development you will likely need at least a scope for verification. Do not assume you only tweaked a couple of register so it should be okay. Small variations can cause very hard to find errors



External Bus Interface - Lab

OPEN: Labs\EBI_demo\template\IAR\SK-s6e2cc_EBI_demo.eww



So what is different? This is just another expansion of the printf example that we have been using

This is another PDL example.
See the Init_PSram (); function in main.c
Grabbed from the “ebif” example.

Main() modified to :

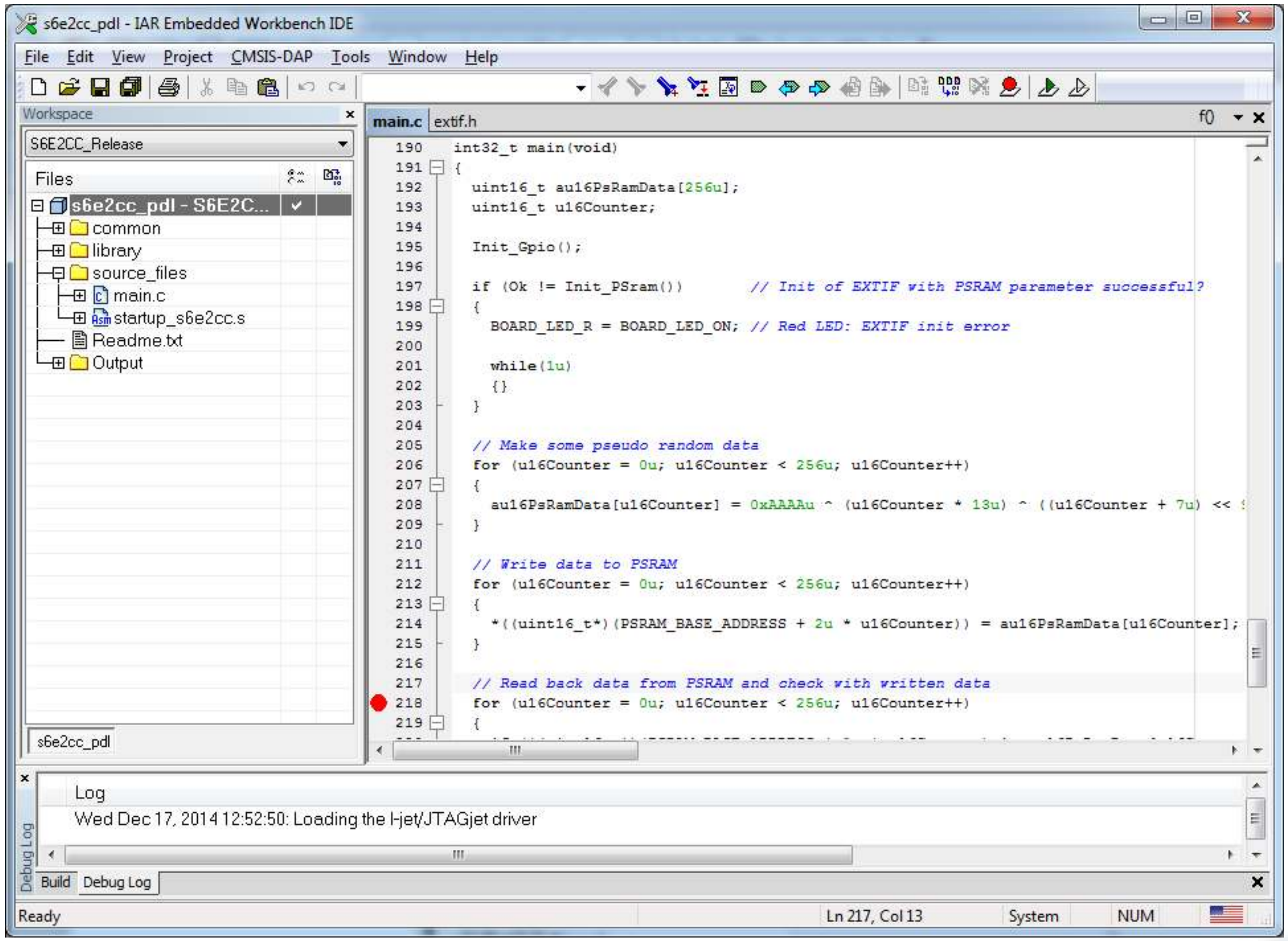
```
call Init_Psram()
create a buffer of pseudo random data
write that data to psram and then verify the data
```

Download and run code – suggest setting breakpoint before the verification routine (about line 216) and messing with the PSRAM memory data in the display memory window. Data area is located at 0x60000000 if the debugger does not load by default.

Running EBI example

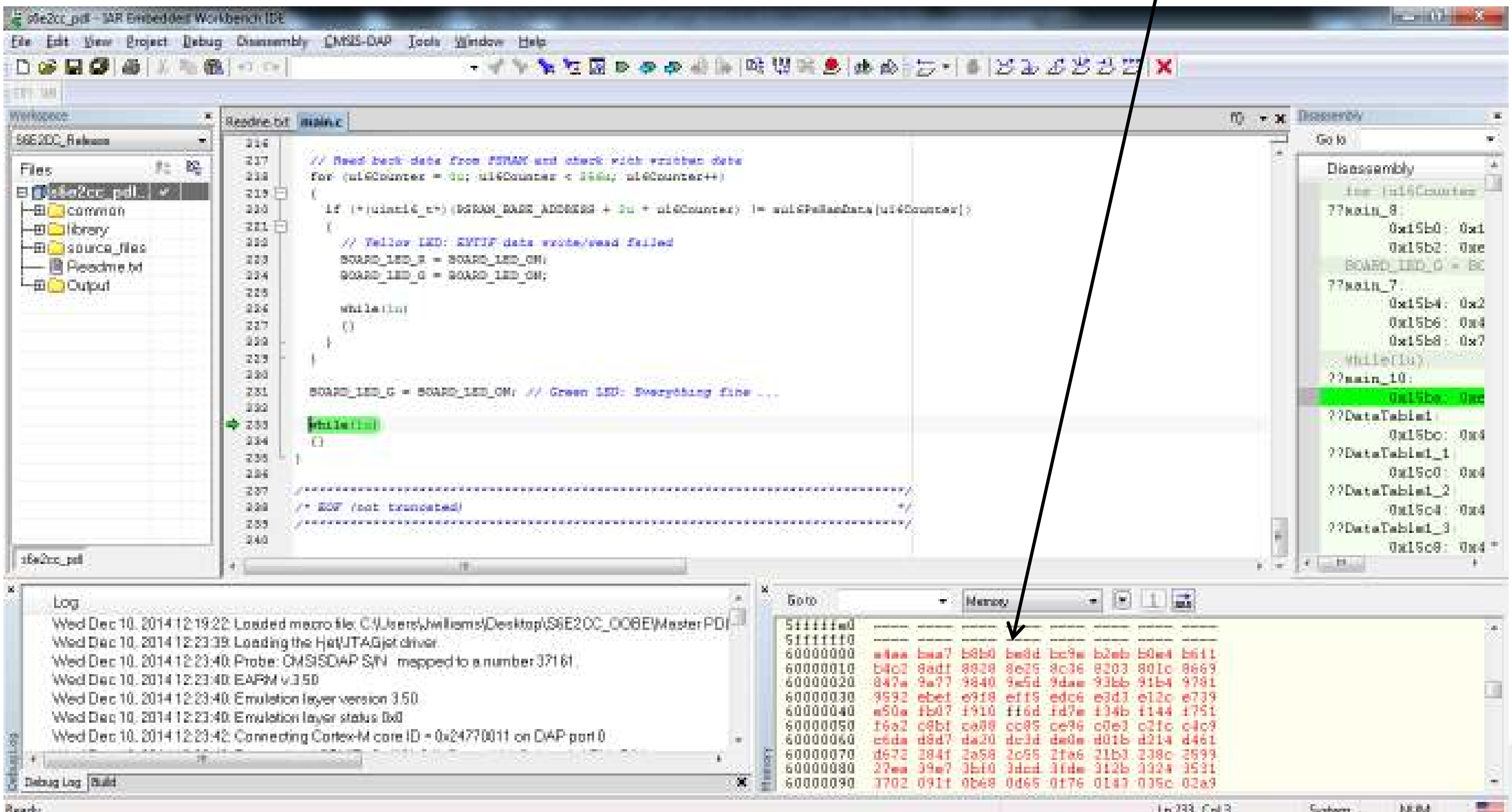
So what is different?

This is a structured PDL example. It passes a structure to the PSRAM_init which probably should be called EBI_Init() as it does much more than enabling the PSRAM



Select project pull down from the title menu and then click rebuild all.
If build is successful launch the debugger with the  button in title bar.

After running the code you can stop and experiment in the memory window peeking and poking the newly available memory.



  CONFIDENTIAL

SPANSION

Conclusion

- We briefly discussed the External Bus Interface Module
- Reviewed the various memory types supported and looked at the access types of each
- Modified printf example to enable and test the external asynchronous interface PSRAM

Inter IC Sound (I^2S) Lecture (including Codec interface)

Purpose and Agenda

Purpose : Implementing I²S on Orion starter kit

- Agenda:
  - I²S block and configuration options
    - Data format
    - Master vs Slave – Clocking setup
  - Wolfson Codec
    - Initialization
    - Streaming data to and from
  - Software
    - Preparation
    - Flow diagrams

FM4 S6E2CC (Orion)

Qualification : Apr/2015

Available Now

■Key function

DSP instruction support, Built-in FPU
2 independent on-chip Flash memories (1MB + 1MB)
Flash 0 wait cycle access up to 200MHz with accelerator
Multi Function Timer & QPRC for 3-phase motor control
Independent power supply terminal for RTC (VBAT)

■Basic specification

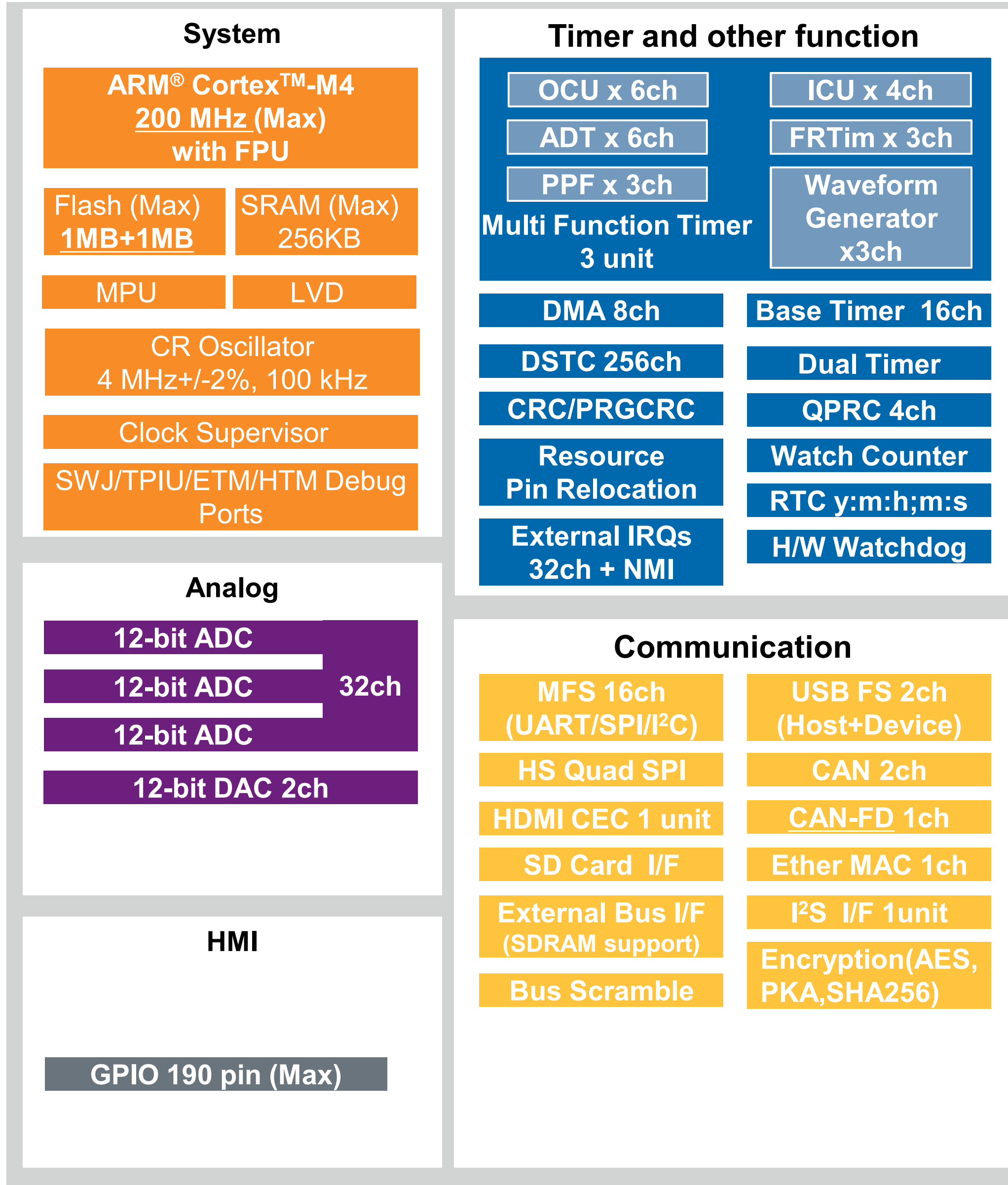
Operating voltage : V_{CC} = 2.7 V to 5.5 V
Operating temperature : T_A = -40˚C to + 105˚C
A/D Conversion time : 2 Msps (0.5 μs)
I²C Standard-mode (100 kbps) / Fast-mode (400 kbps) /
Fast mode plus (1000 kbps)
Unique ID : 41bits

■Power consumption

Operation current : 0.4 mA/ MHz (CPU only)
RTC operation current (VBAT) : 1.0 μA
6 Low-power consumption modes supported
(SLEEP,TIMER,RTC,STOP,DS-RTC,DS-STOP)

Part number	Flash Memory	RAM
S6E2CCAxx	2MB	256KB
S6E2CC9xx	1.5MB	192KB
S6E2CC8xx	1MB	128KB

Part number	Package
S6E2CCxHx	LQFP 144 pins (0.5 mm pitch)
S6E2CCxJx	LQFP 176 pins (0.5 mm pitch)
S6E2CCxLx	LQFP 216 pins (0.5 mm pitch)
S6E2CCxJx	BGA 192 pins (0.8 mm pitch)



What is I^2S

I^2S

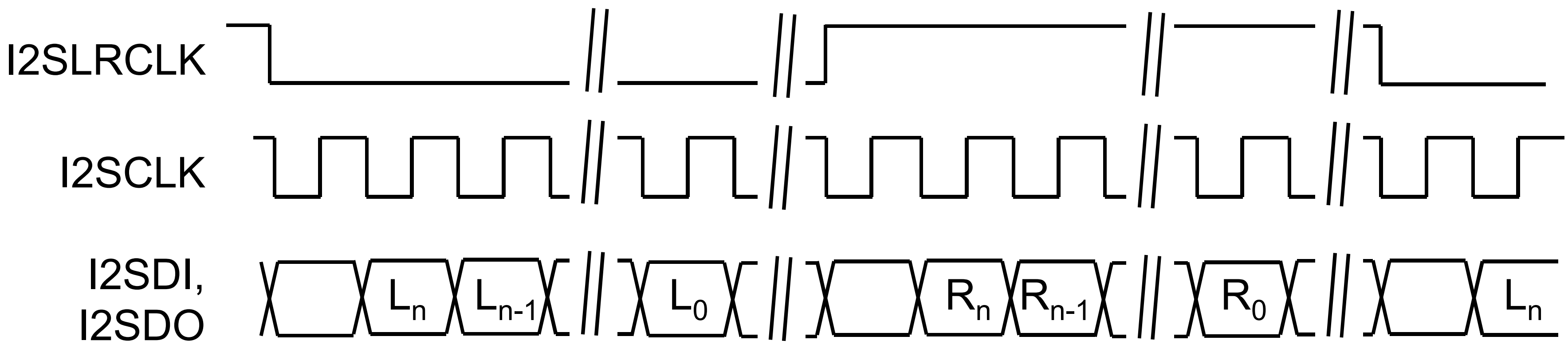
- What is I^2S?
  - I^2S stands for Inter-IC Sound.
  - I^2S was introduced in 1986 by Philips. Latest revision was in 1996.
  - I^2S is a serial bus interface standard for connecting digital audio devices.
  - I^2S uses PCM (Pulse Code Modulation) data as it is, so it uses no compression like MP3.
  - I^2S uses up to 5 lines.
    - Bit clock (I2SCLK)
    - Word select/strobe (I2SWS / I2SLRCLK)
    - Data in (I2SDI) [1]
    - Data out (I2SDO) [1]
    - Master clock (I2SMCLK) [2]

[1] At least one of these lines has to be provided
[2] Optional, if codec is slave and needs higher clock than I2SCLK

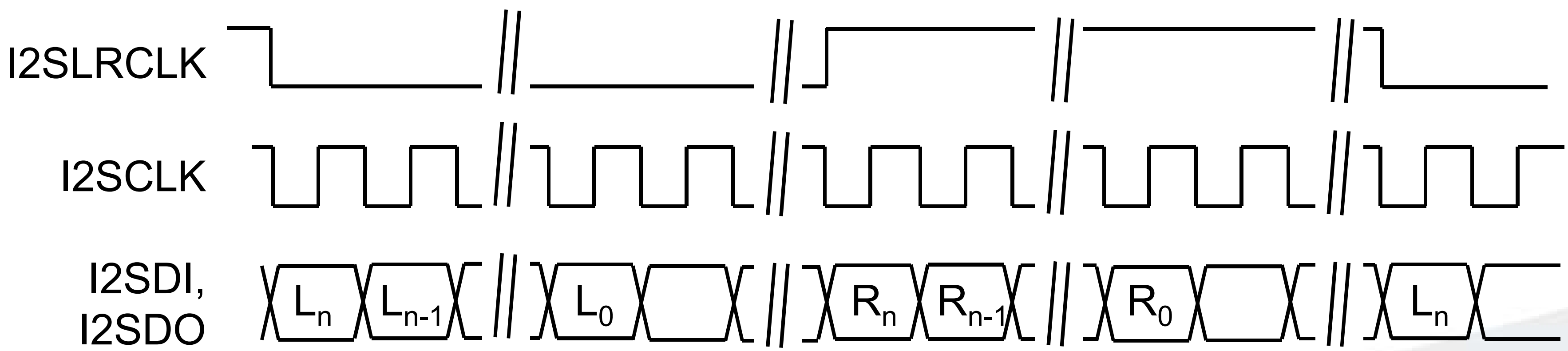
I²S Formats (1)

I²S Format

- I²S Format (MSB first*)



- Left Justified Format (MSB first)

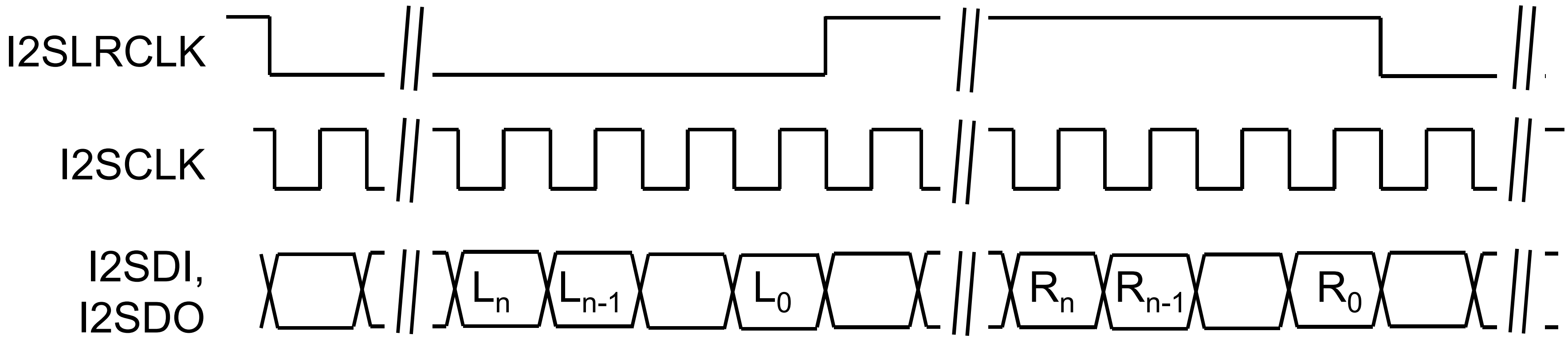


n = Bit Length

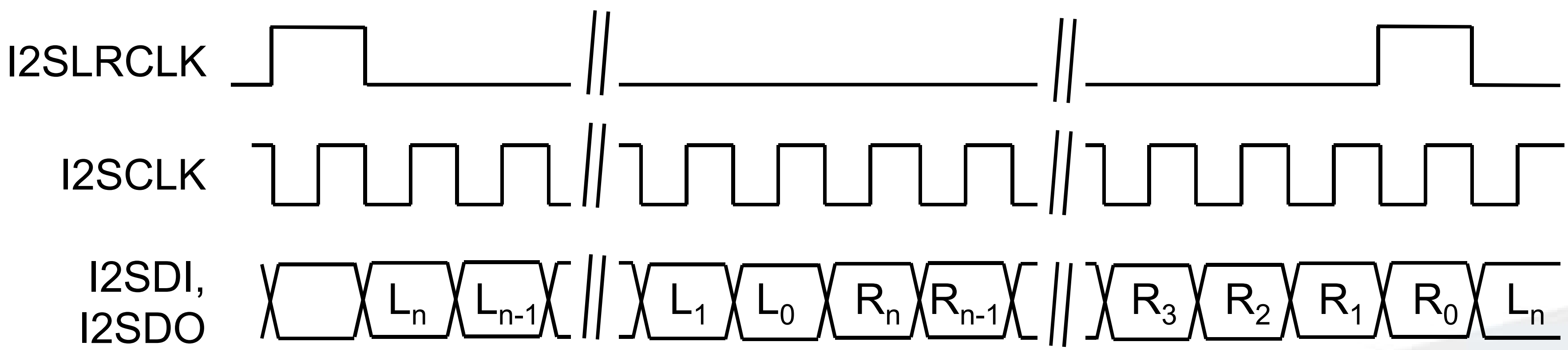
*

I^2S Format

- Right Justified Format (MSB first)



- PCM Format A (MSB first)

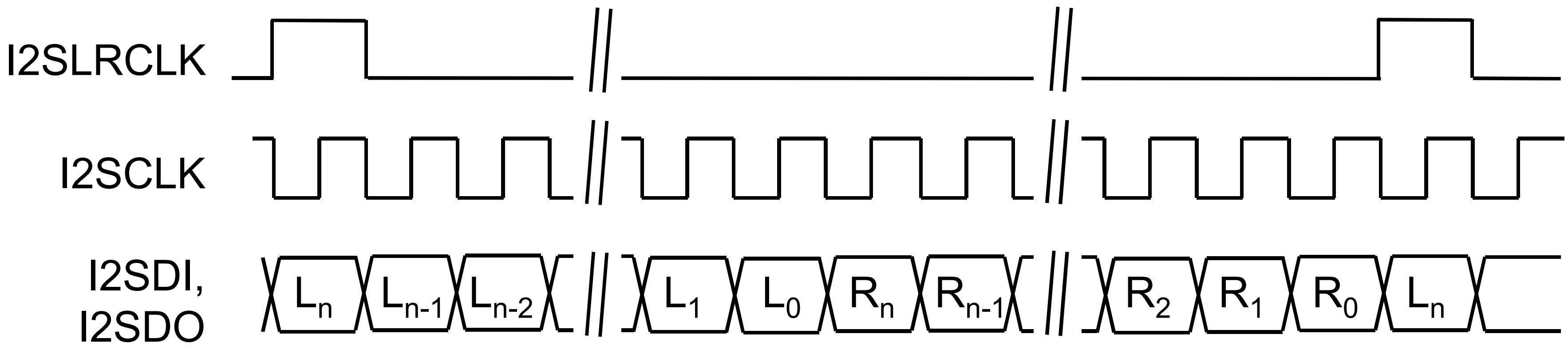


n = Bit Length

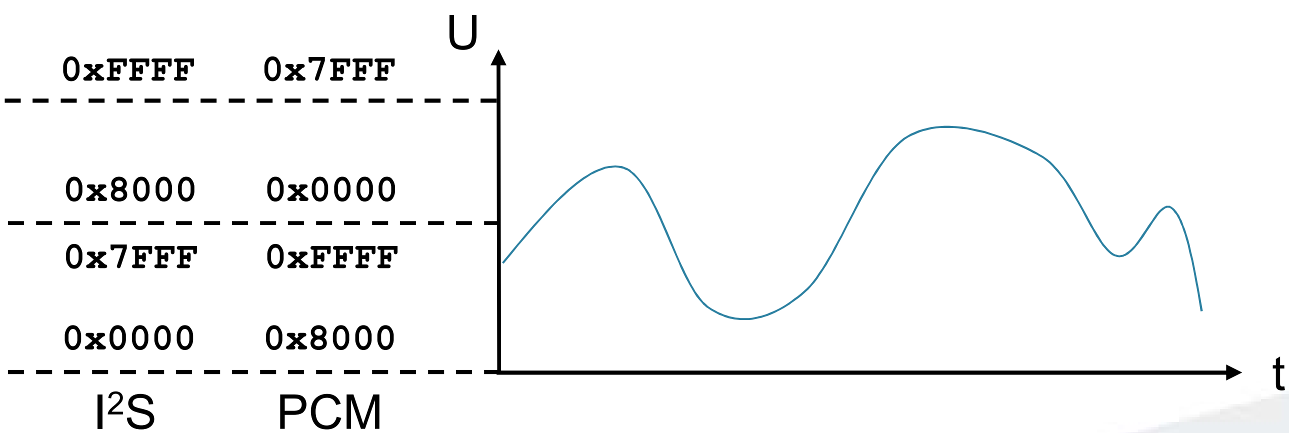
I²S Formats (3)

I²S Format

- PCM Format B (MSB first)

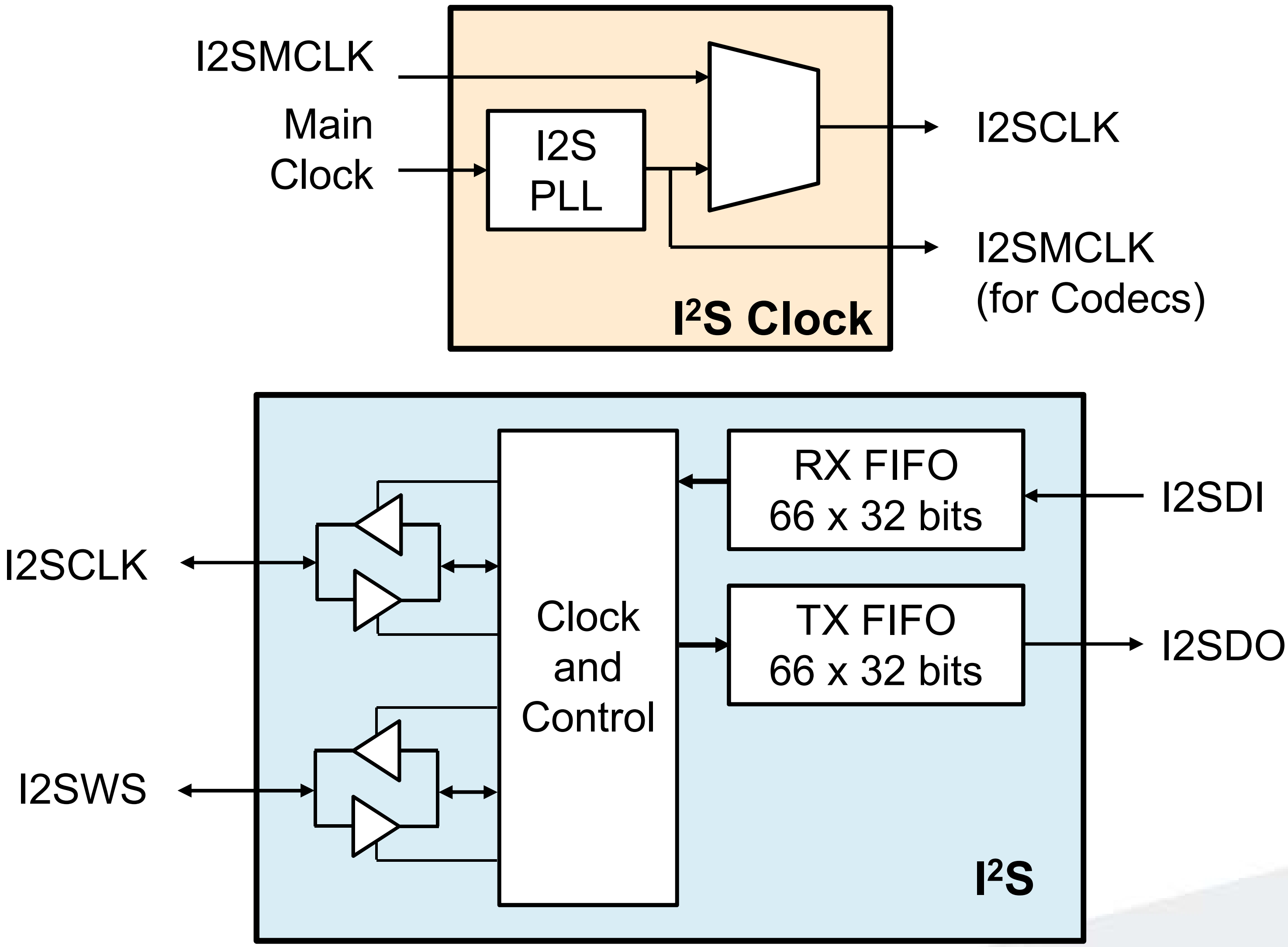


- Data Format (here: 16 bits)



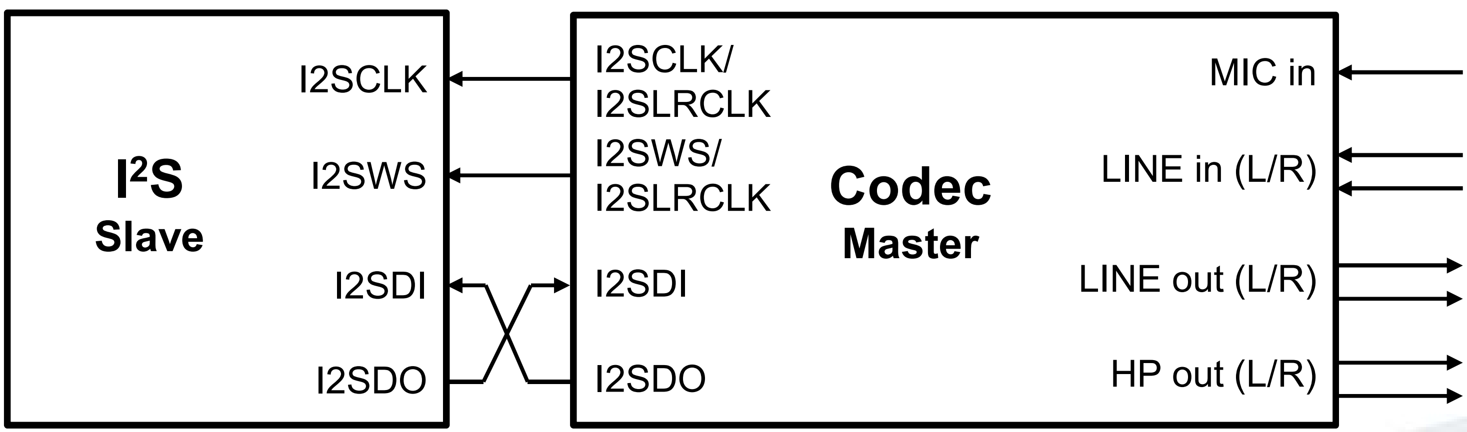
Spansion I²S Block on Orion

I²S and I²S Clock



I^2S Master/Slave Mode Clocking

- For using the Orion's I^2S with standard sample rates (e.g. 44.1 kHz or 48 kHz) an external crystal of 19.2 MHz is needed.
- The Orion boards (SK-FM4-216-ETHERNET and SK-FM4-176L-S6SE2CC) are equipped with an external 4 MHz crystal.
  - With 4 MHz Main Clock, no standard I^2S sample rates can be generated.
  - Therefore, the I^2S is switched to slave mode, and the external codec generates the I^2S clock (I2SCLK/I2SLRCLK).

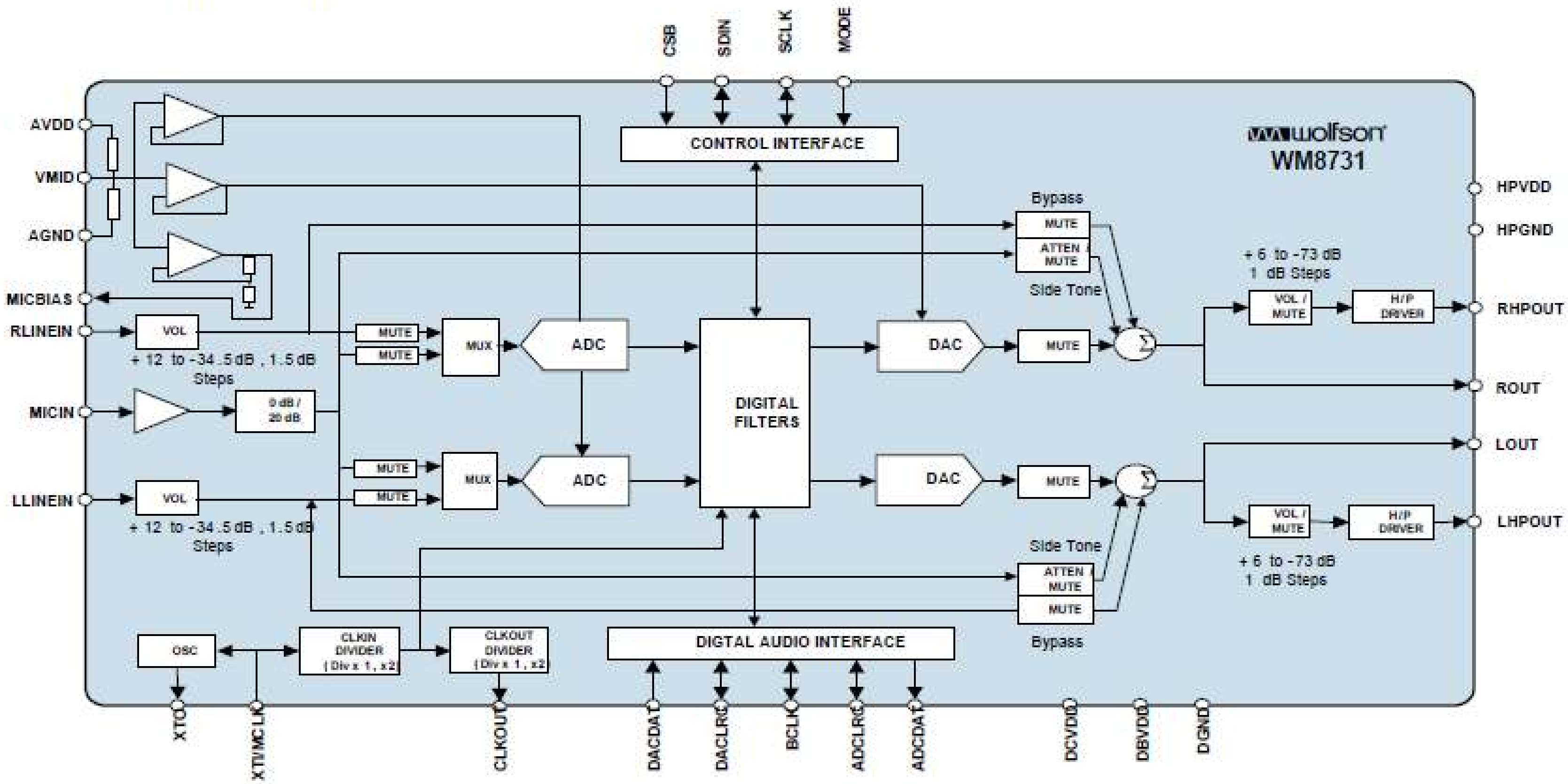


The word codec is a portmanteau of "coder-decoder"

Wolfson Codec – WM8731

Wolfson Codec – WM8731

BLOCK DIAGRAM



CODER DECODER

Specification overview

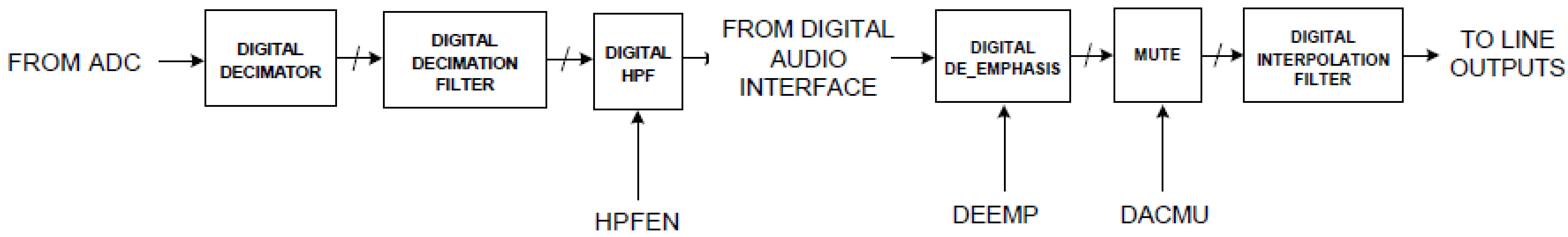
Test Conditions

AVDD, HPVDD, DBVDD = 1.8V, AGND = 0V, DCVDD = 1.5V, DGND = 0V, T_A = +25°C, Slave Mode, fs = 48kHz, XTI/MCLK = 256fs unless otherwise stated.

Stereo Headphone Output						
0dB Full scale output voltage				1.0 x AVDD/3.3		Vrms
Max Output Power	P_O	RL = 32 Ω		9		mW
		RL = 16 Ω		18		
Signal to Noise Ratio (Note 1,3)	SNR	A-weighted	86	95		dB
Total Harmonic Distortion	THD	1kHz, -5dB FS signal R_L = 32Ω		0.08 -62	0.1 -60	% dB
		1kHz, -2dB FS signal R_L = 32Ω			1 -40	
Power Supply Rejection Ratio	PSRR	1kHz 100mVpp		50		dB
		20Hz - 20kHz, 100mVpp		45		
Programmable Gain		1kHz	-73	0	6	dB
Programmable Gain Step Size		1kHz		1		dB
Mute attenuation		1kHz, 0dB		80		dB
Microphone Input to Headphone Output Side Tone Mode						
0dB Full scale output voltage				1.0 x AVDD/3.3		Vrms
Signal to Noise Ratio (Note 1,3)	SNR		85	90		dB
Power Supply Rejection Ratio	PSRR	1kHz 100mVpp		50		dB
		20Hz to 20kHz 100mVpp		45		
Programmable Attenuation		1kHz	6		15	dB
Programmable Attenuation Step Size		1kHz		3		dB
Mute attenuation		1kHz, 0dB		80		dB

Wolfson Codec – WM8731

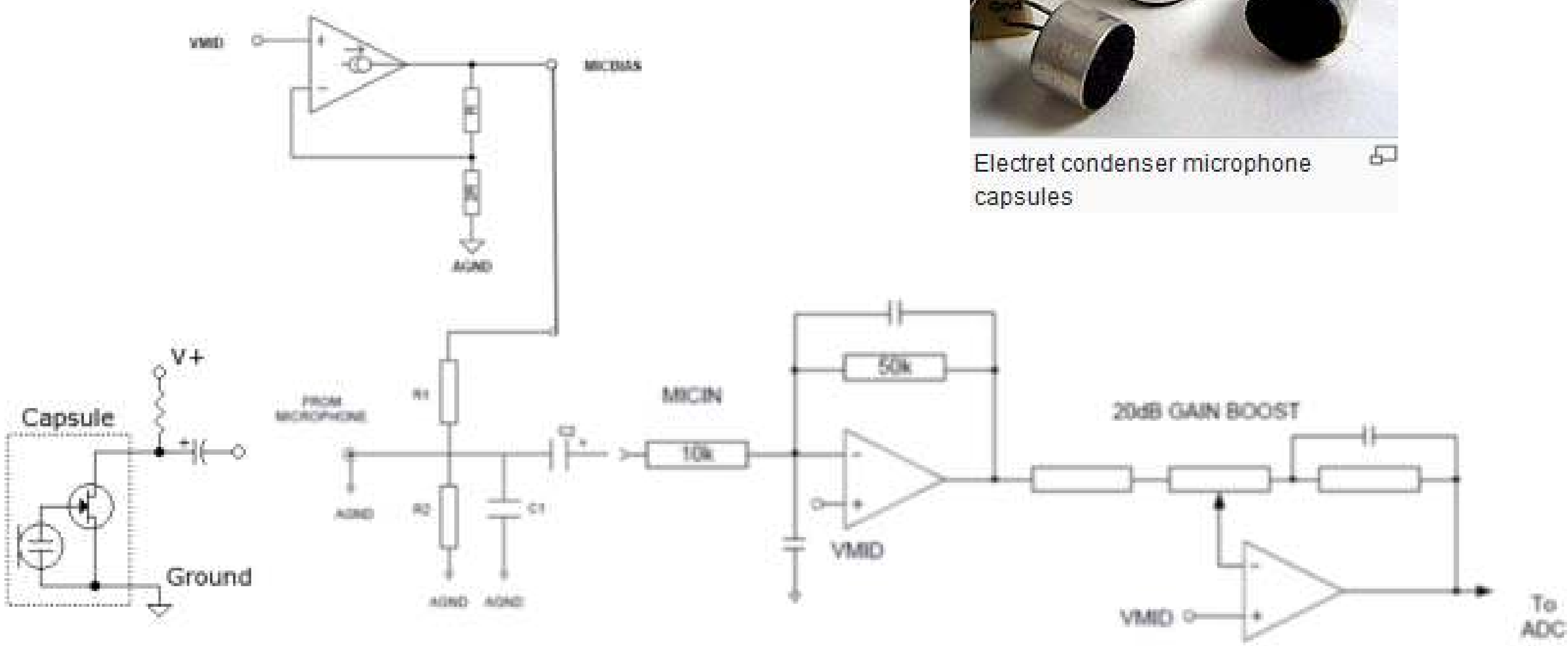
On-chip digital filtering



Wolfson Codec – WM8731

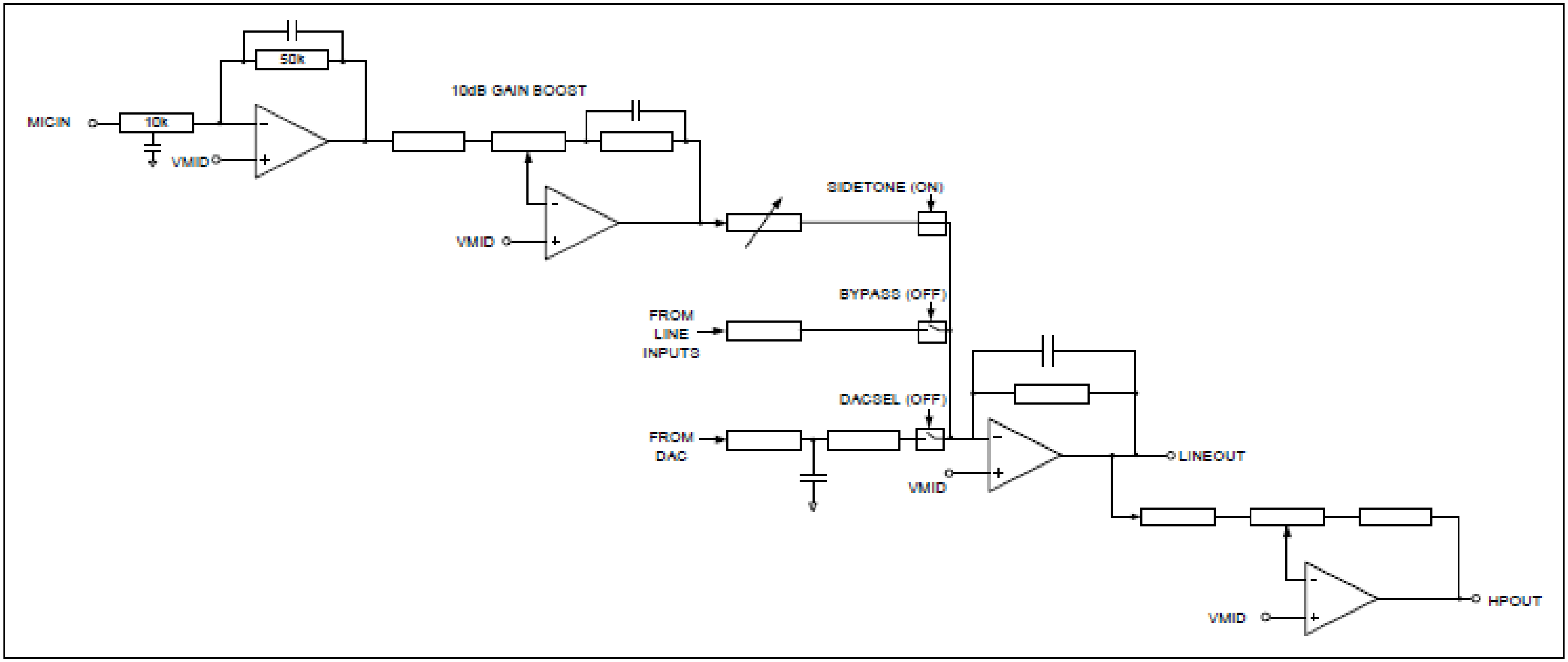
Microphone Ready

Electret condenser microphone capsules



** Vmid is mid-rail decoupling

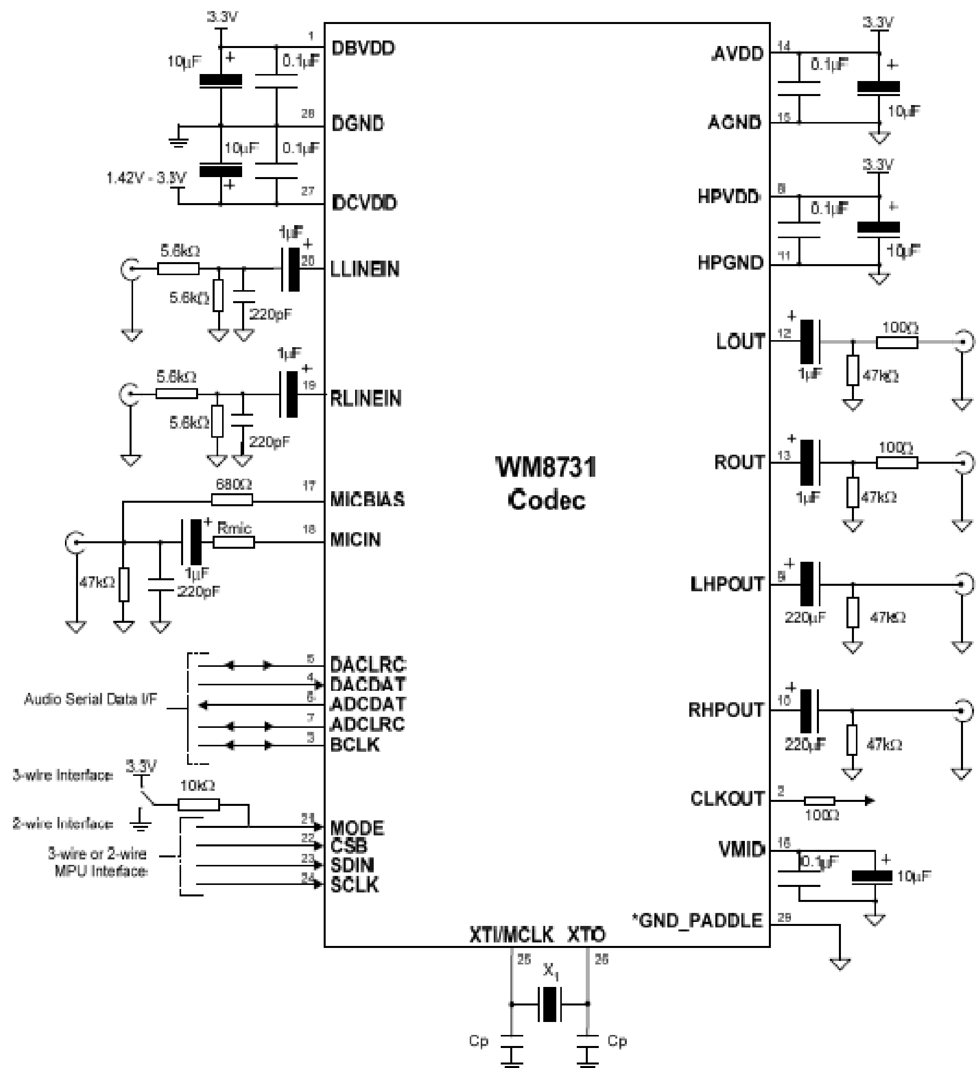
Headphone out



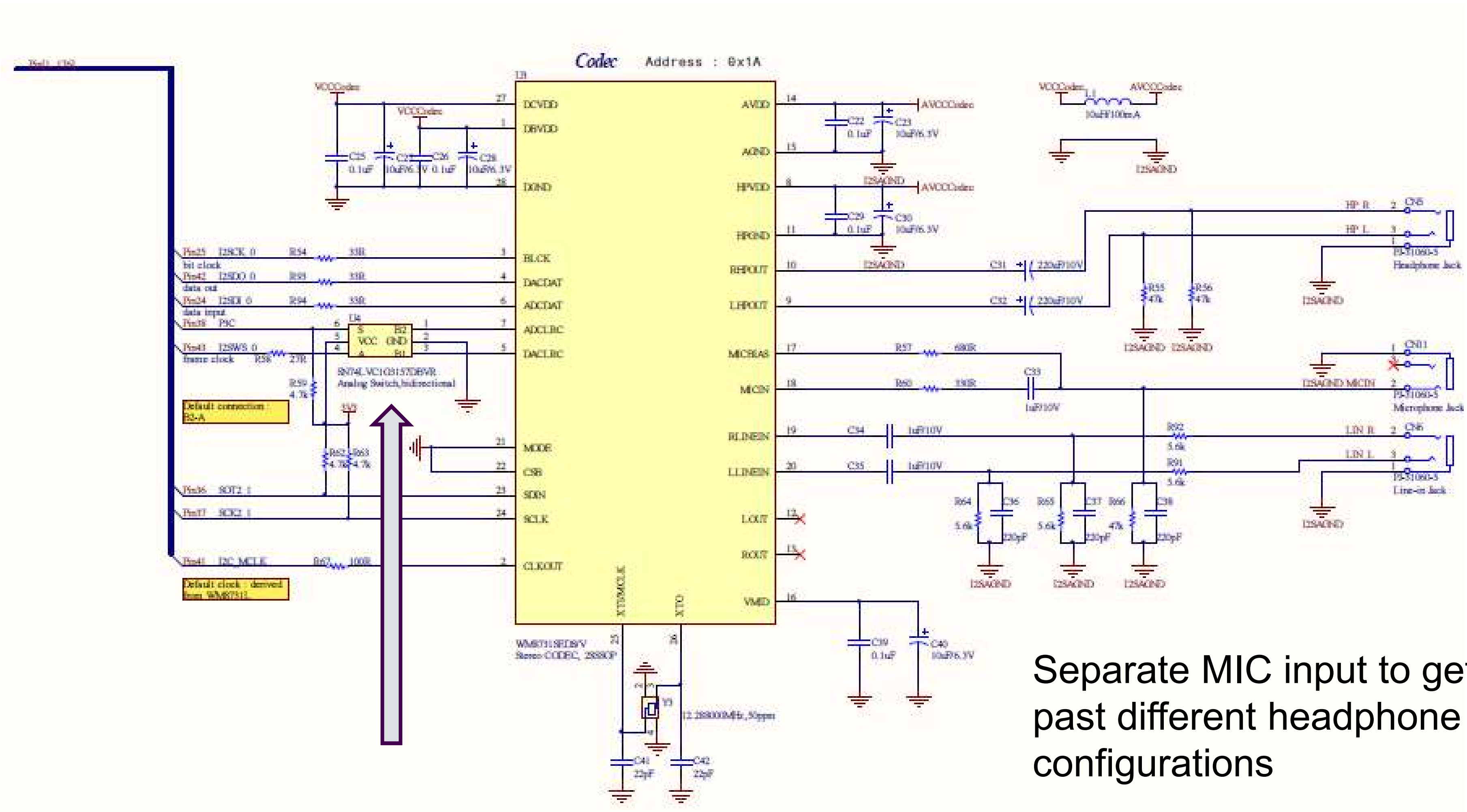
WM8731 external component connections.

RECOMMENDED EXTERNAL COMPONENTS

Stereo 24-bit multi-bit sigma delta ADCs and DACs are used with oversampling digital interpolation and decimation filters. Digital audio input word lengths from 16-32 bits and sampling rates from 8 kHz to 96 kHz are supported.



Orion starter kit codec connections



Separate MIC input to get past different headphone configurations

ADC / DAC clock select possibility

CONFIDENTIAL

SPANSION

REGISTER MAP

The complete register map is shown in Table 29. The detailed description can be found in Table 30 and in the relevant text of the device description. There are 11 registers with 16 bits per register (7 bit address + 9 bits of data). These can be controlled using either the 2 wire or 3 wire MPU interface.

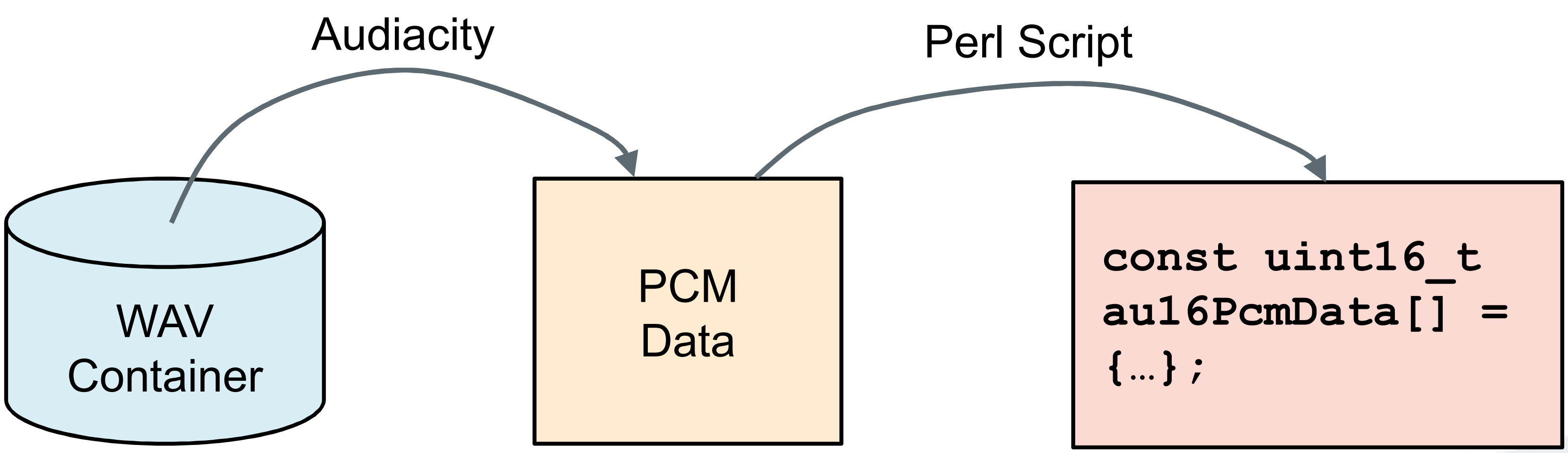
REGISTER	BIT[8]	BIT[7]	BIT[6]	BIT[5]	BIT[4]	BIT[3]	BIT[2]	BIT[1]	BIT[0]	DEFAULT
R0 (00h) Left Line In	LRINBOTH	LINMUTE	0	0	LINVOL[4:0]					0_1001_0111
R1 (01h) Right Line In	RLINBOTH	RINMUTE	0	0	RINVOL[4:0]					0_1001_0111
R2 (02h) Left Headphone Out	LRHPBOTH	LZCEN	LHPVOL[6:0]							0_0111_1001
R1 (01h) Right Headphone Out	RLHPBOTH	RZCEN	RHPVOL[6:0]							0_0111_1001
R4 (04h) Analogue Audio Path Control	0	SIDEATT[1:0]		SIDETONE	DACSEL	BYPASS	INSEL	MUTEMIC	MICBOOST	0_0000_1010
R5 (05h) Digital Audio Path Control	0	0	0	0	HPOR	DACMU	DEEMPH[1:0]		ADCHPD	0_0000_1000
R6 (06h) Power Down Control	0	POWEROFF	CLKOUTPD	OSCPD	OUTPD	DACPD	ADCPD	MICPD	LINEINPD	0_1001_1111
R7 (07h) Digital Audio Interface Format	0	BCLKINV	MS	LRSWAP	LRP	IWL[1:0]		FORMAT[1:0]		0_1001_1111
R8 (08h) Sampling Control	0	CLKODIV2	CLKIDIV2	SR[3:0]				BOSR	USB/ NORMAL	0_0000_0000
R9 (09h) Active Control	0	0	0	0	0	0	0	0	Active	0_0000_0000
R15 (0Fh) Reset	RESET[8:0]									not reset

Test Software written for SK-FM4-176L-S6SE2CC

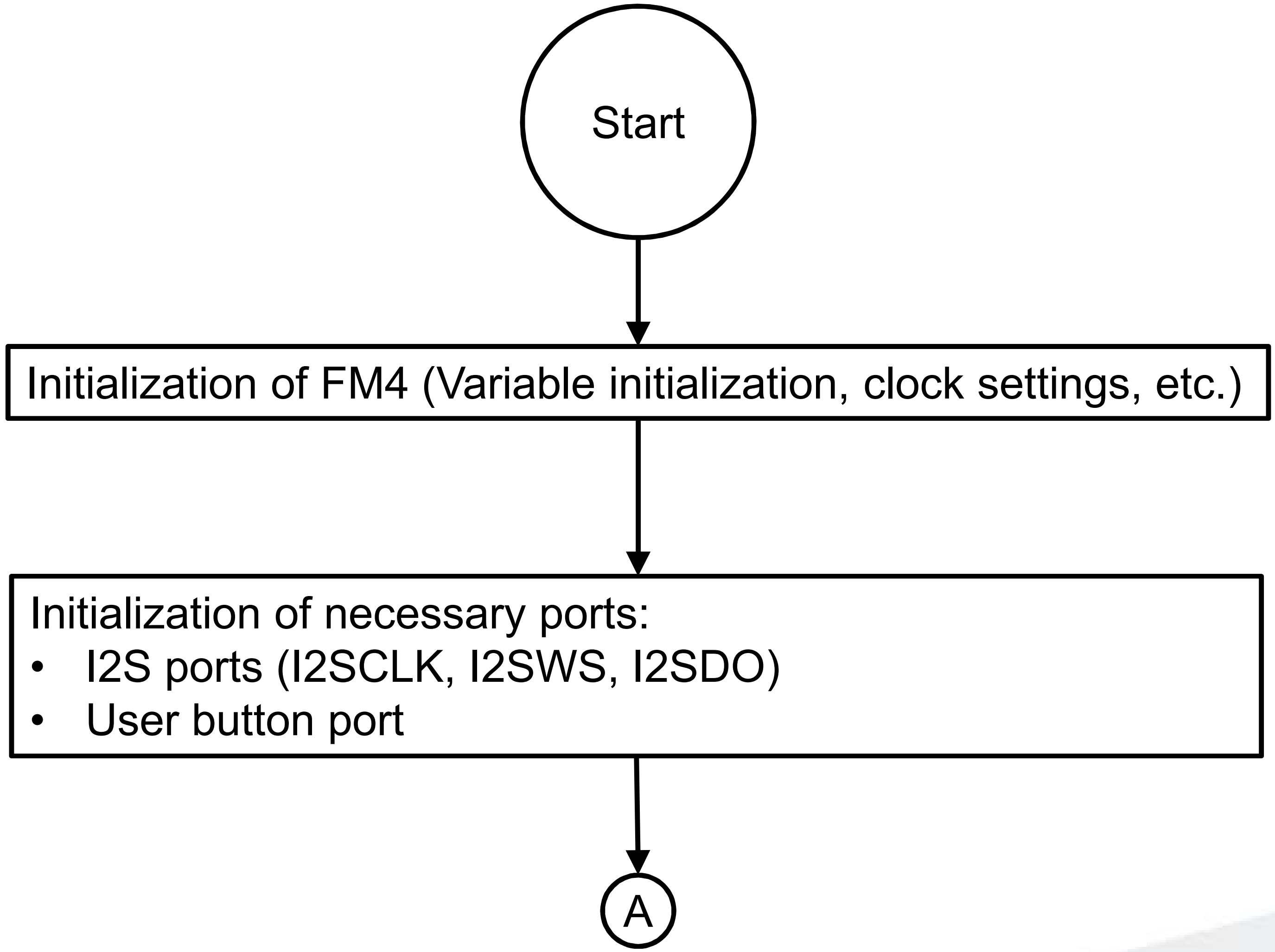
Sound Output Software

Preparation of Software used for Sound Output

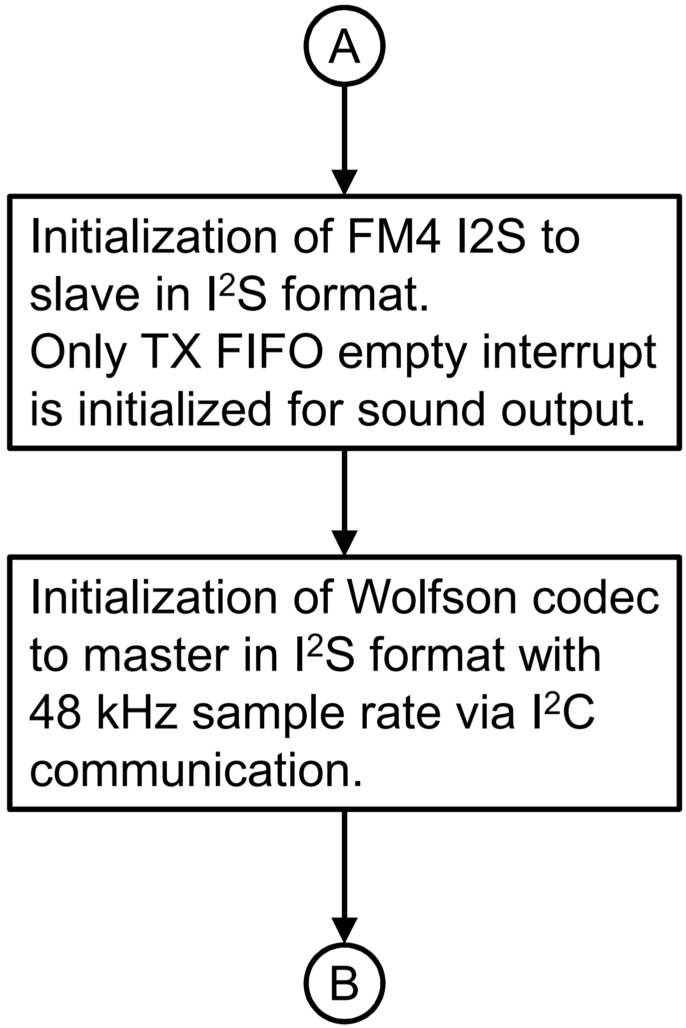
- Preparation
  - A short sound file in WAV format was taken and transformed with an open source audio editor (Audiacity) into PCM data.
  - A Perl script was used to generate a C module with a const array from the PCM data.
  - This const array was put into the project.



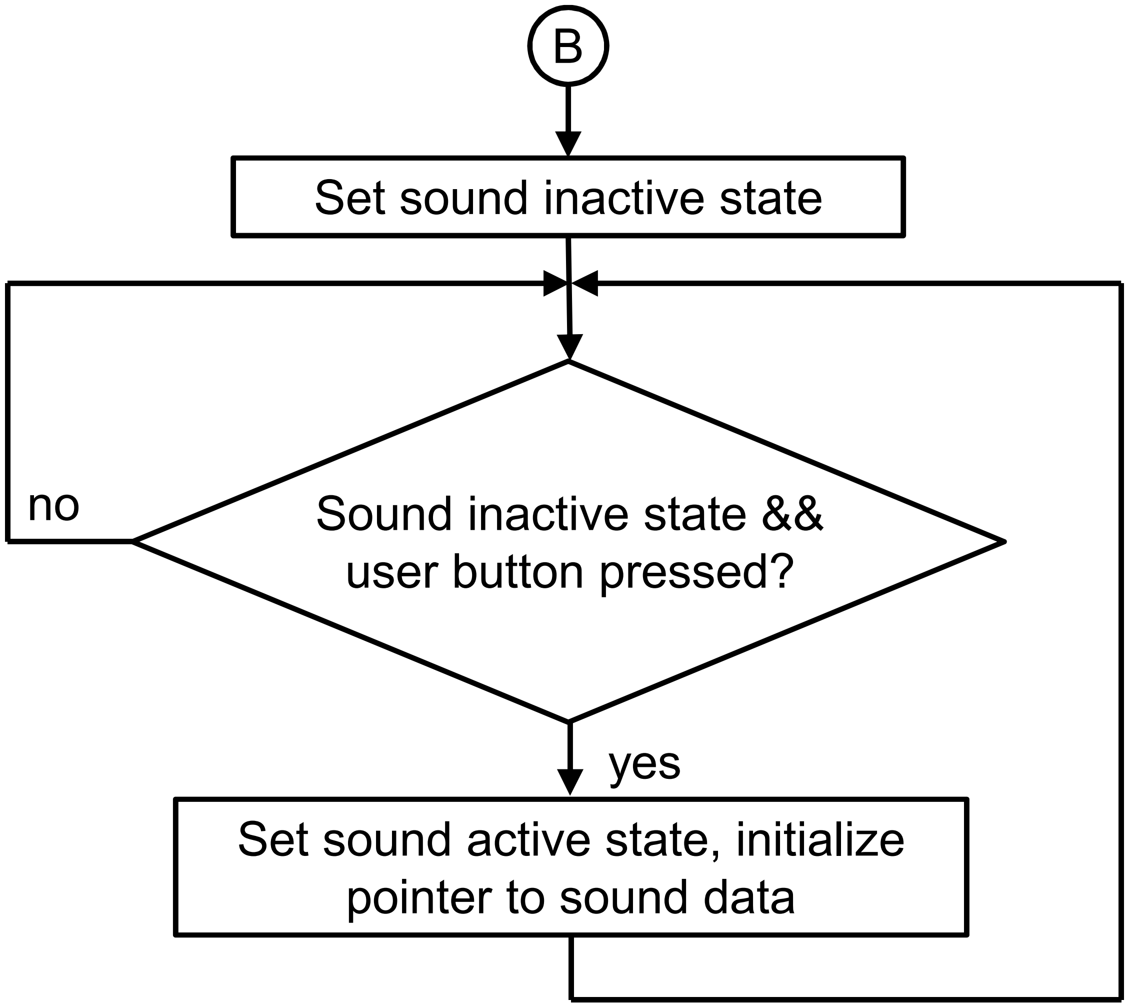
Software used for Sound Output (1)



Software used for Sound Output (2)

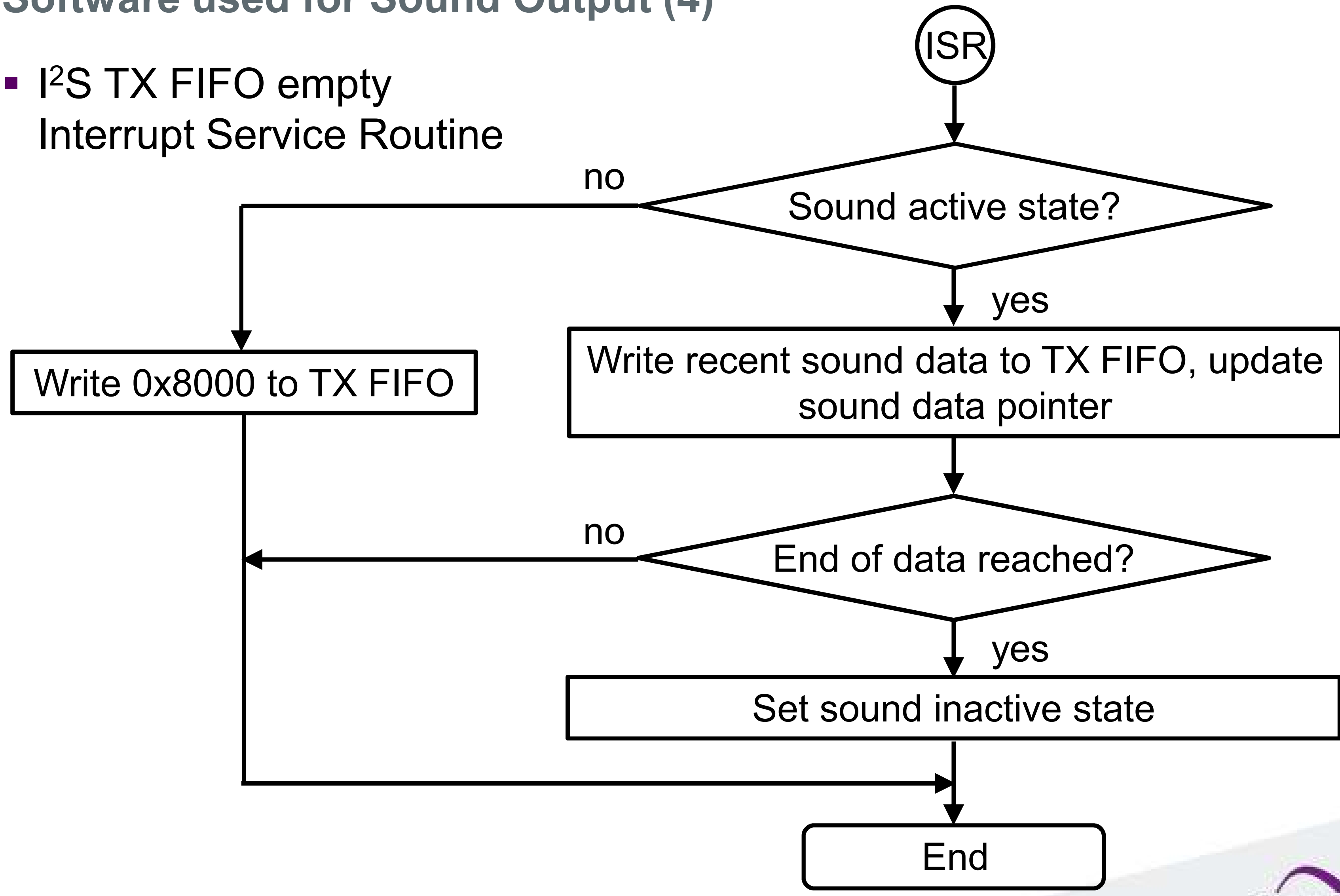


Software used for Sound Output (3)



Software used for Sound Output (4)

- I^2S TX FIFO empty
  Interrupt Service Routine



SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Dual Flash / Programming via RAM

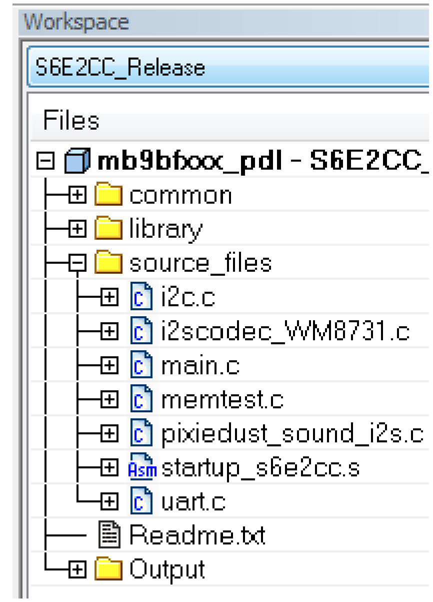
Lab - SK-FM4-176L-S6SE2CC

I2S device interfacing using the on board CODEC

I²S and Codec - Lab

OPEN: Labs\i2s_sound\SK-s6e2cc_i2s_sound-V12\example\IAR\s6e2cc_i2s.eww

So what is different?



This is basically the same PDL example with added routines in function main() used with I²S routines.

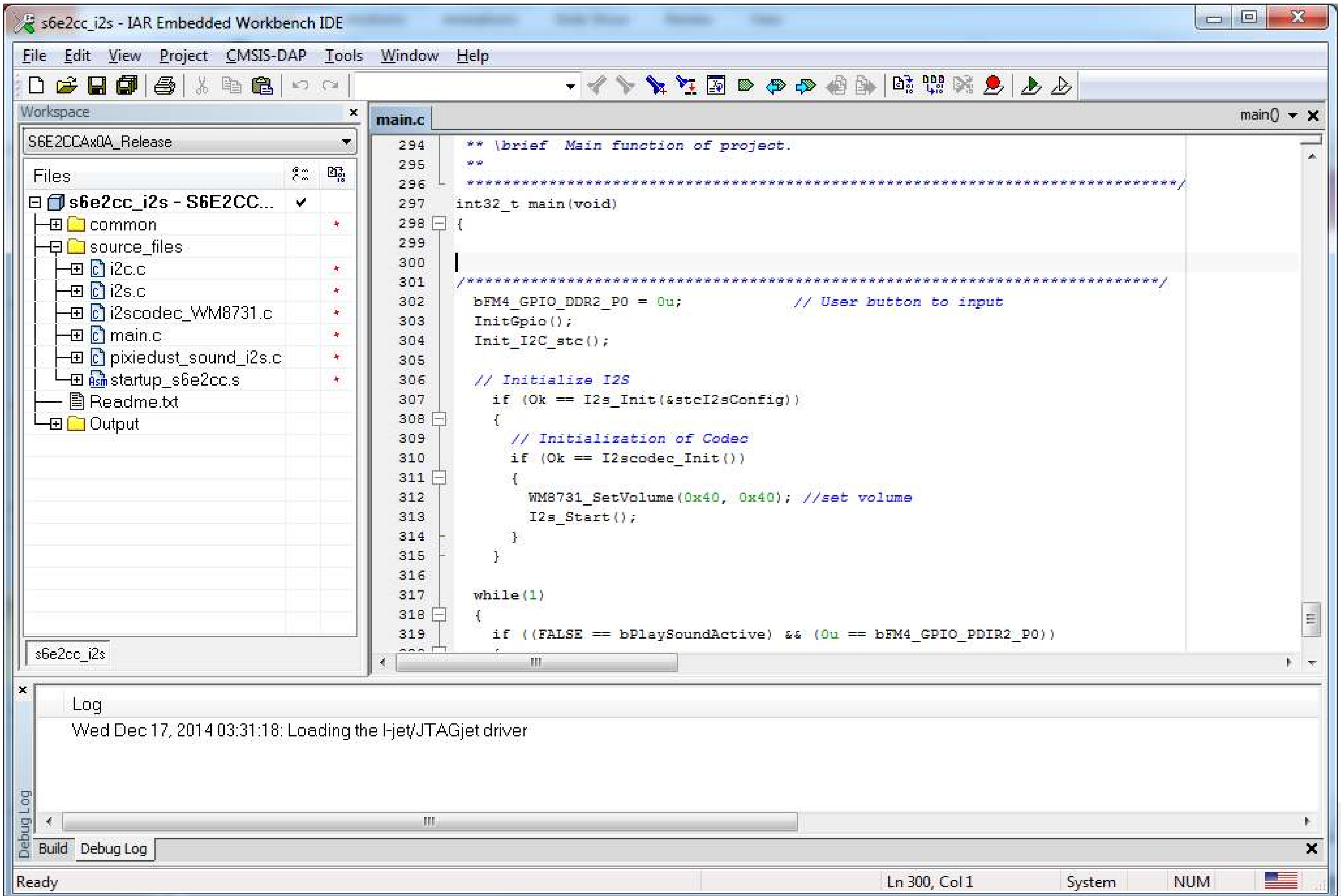
Main() modified to :
`init_i2s()`

Download and run code –

Select project pull down from the title menu and then click rebuild all.
If build is successful launch the debugger with the  button in title bar.

  CONFIDENTIAL

I2S and CODEC



I2S and CODEC

- Once code is executing a user button press will trigger the a message or tone to be outputted from the I2S to the CODEC
- Due to a manufacturing issue with the headphone jack you may have to jostle the plug slightly to hear the output and then might only hear left or right one at a time. (this is only an issue with these 20 prototype units. No production units will have this issue)

Conclusion

- We discussed the Inter-IC sound Interface Module.
- Reviewed the various types of transfers
- Ran an example including the I2S PDL structure and played a tune from headphone speakers.

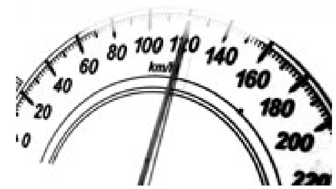
SK-FM4-176L-S6SE2CC List of Labs

- Lab - Quick Start Guide - Board Test Software
- Lab - Flash Programming Tools
- Lab - IAR MCU Template UART and PDL C printf
- Lab - ADC Example Using the Light Sensor
- Lab - Demonstration of EBI code
- Lab - I2S device interfacing using the on board CODEC
- Lab - Dual Flash / Programming via RAM

Lab - Dual Flash / Flash Accelerator

Embedded Flash Memory

- Wide operation single voltage range: [1.65 V] 1.8 V to 5.5 V
- High speed: FM4: 200 MHz
- Embedded programming algorithm
  - Simplifies flash programming within application



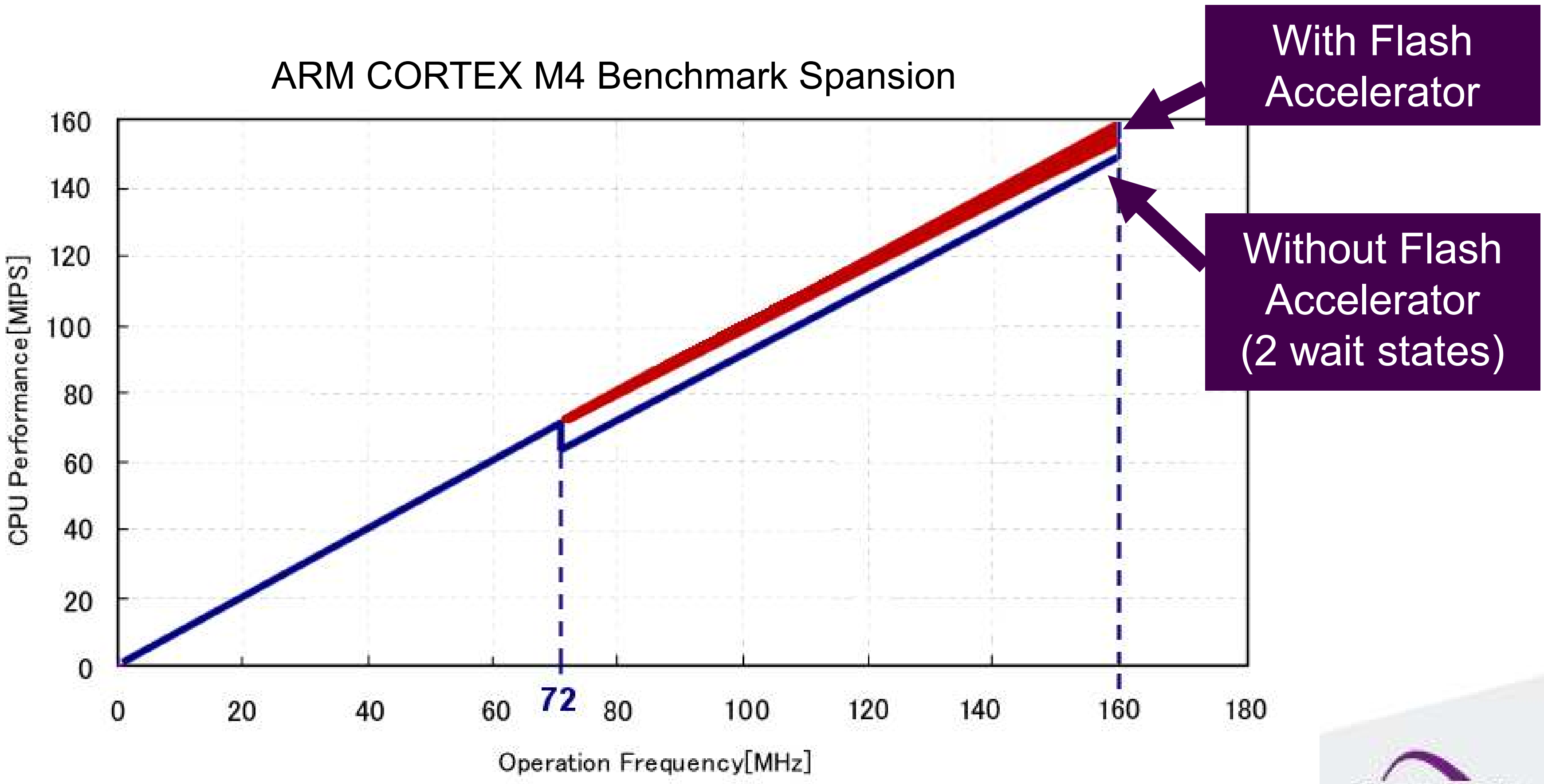
- Split into different sector sizes
- Reliability & quality
  - Same technology for automotive and consumer applications
  - Erase/program cycle: 100,000 times
  - Data retention time: 20 years
  - ECC (detect and correct 1 bit one each word)

- Security
  - Security function for code protection

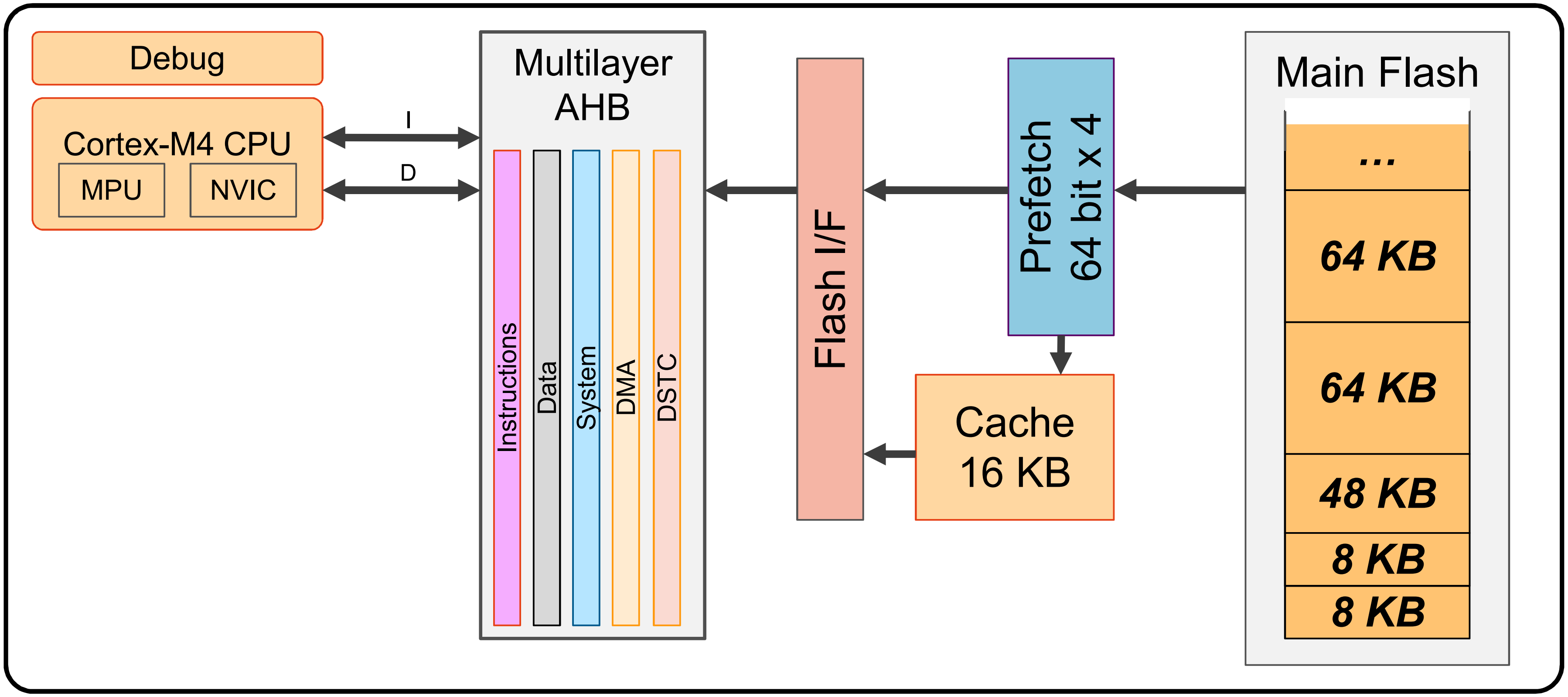
SPANSION

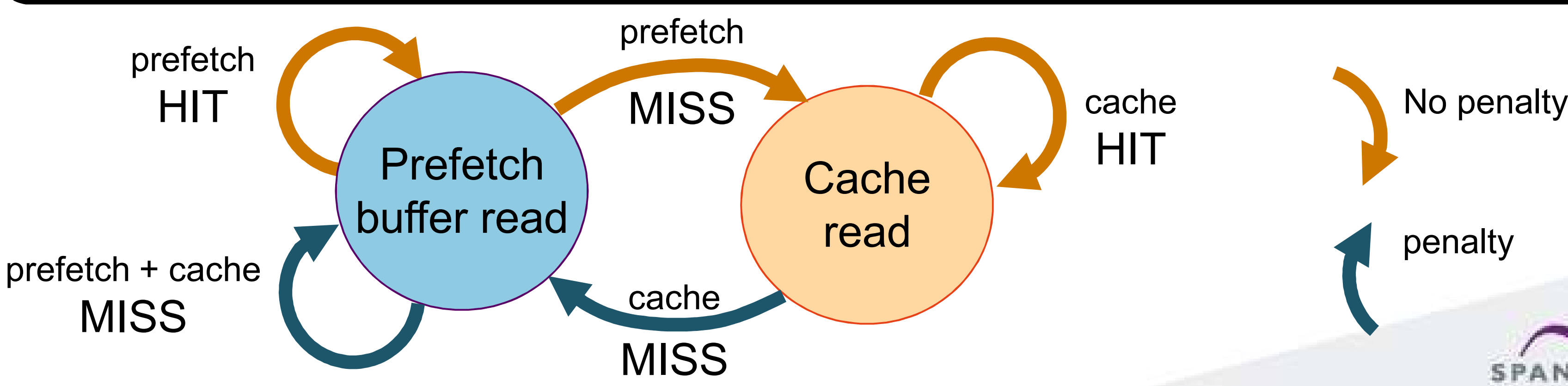
High Performance FM4 (200 MHz)

- High performance operation by Spansion High-Speed Flash
  - 72 MHz – w/o wait states
  - Flash Accelerator delivers “almost-zero-wait-states” Flash Access



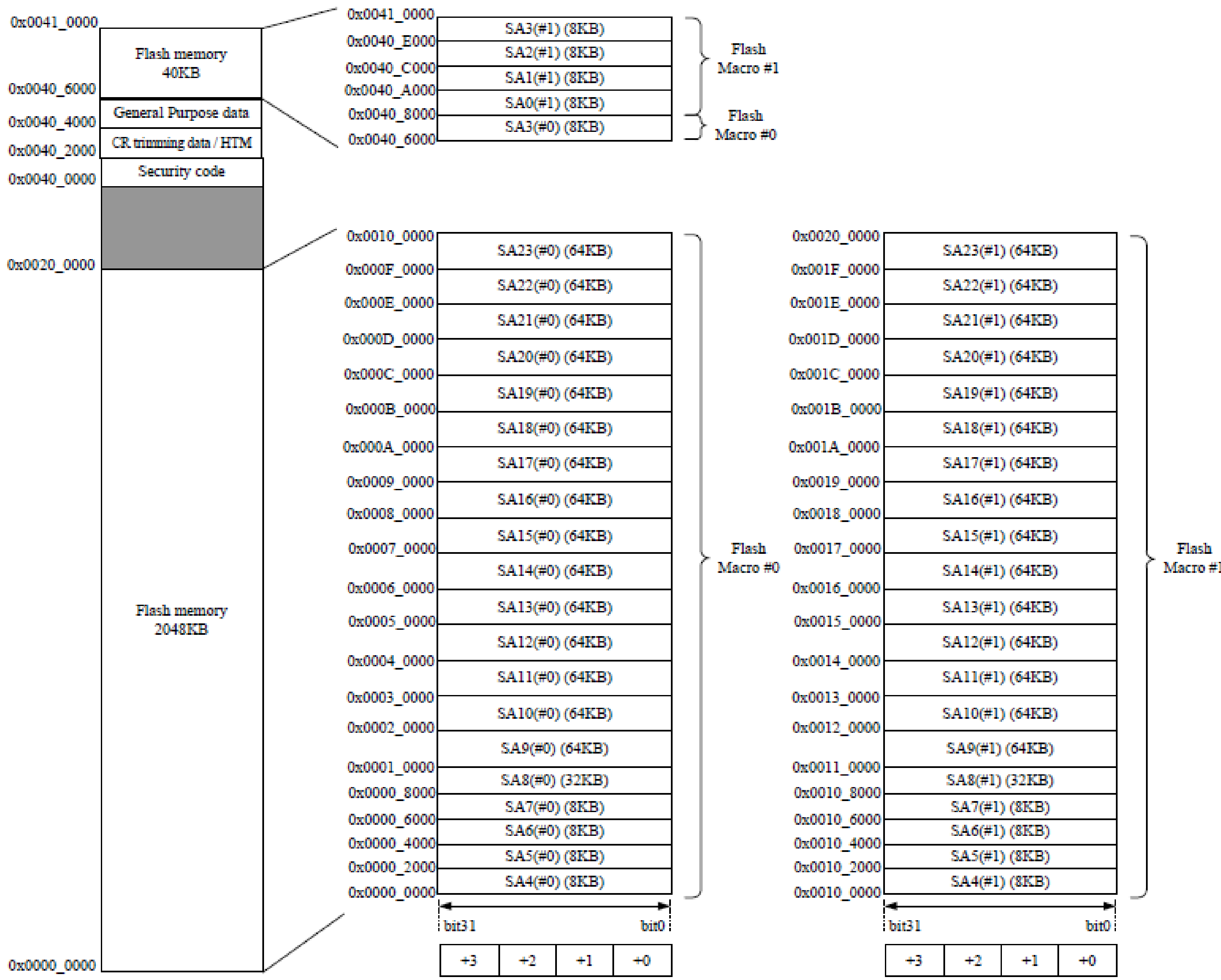
Flash Accelerator



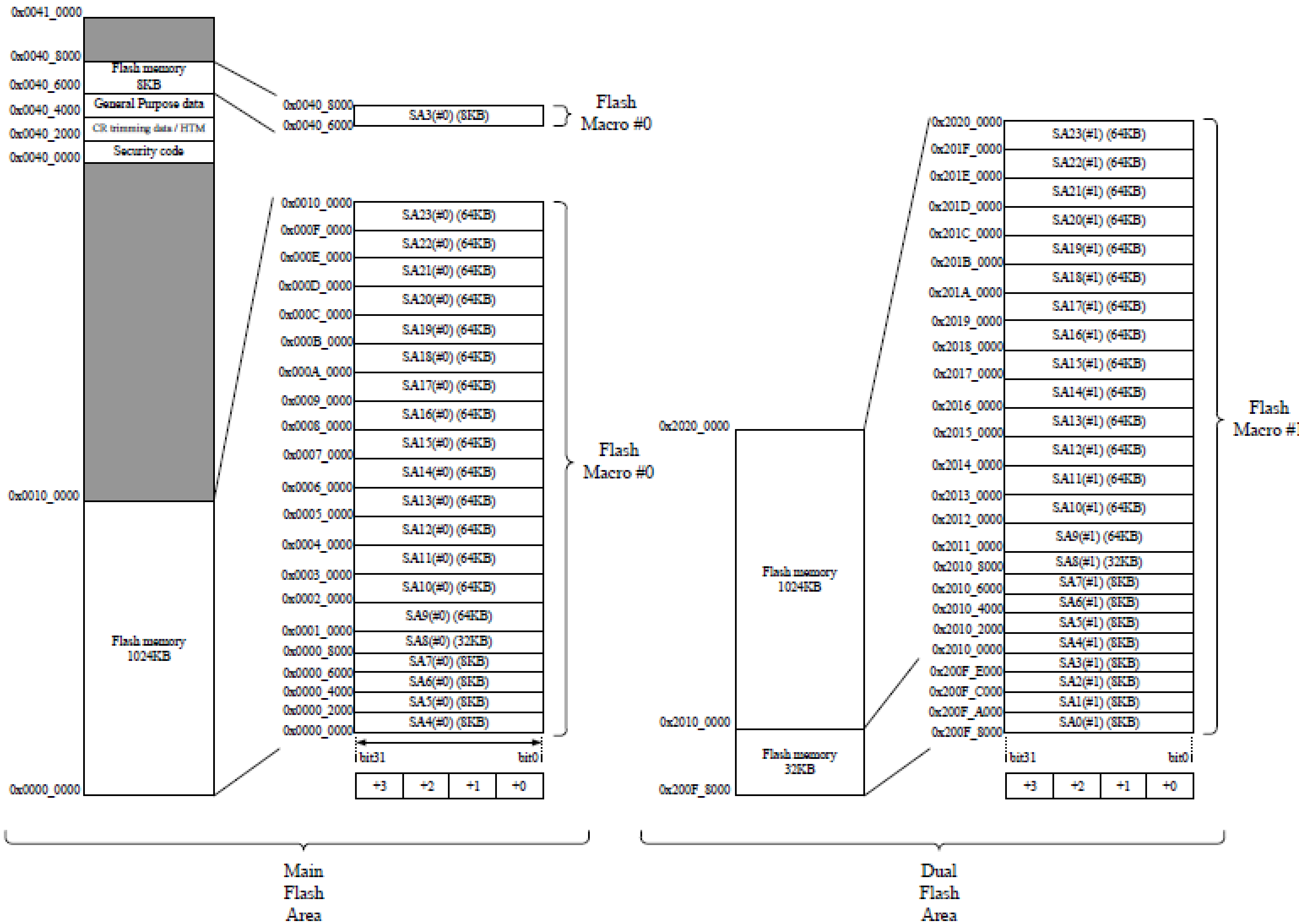


FM4 Flash Memory (Dual Operation Flash)

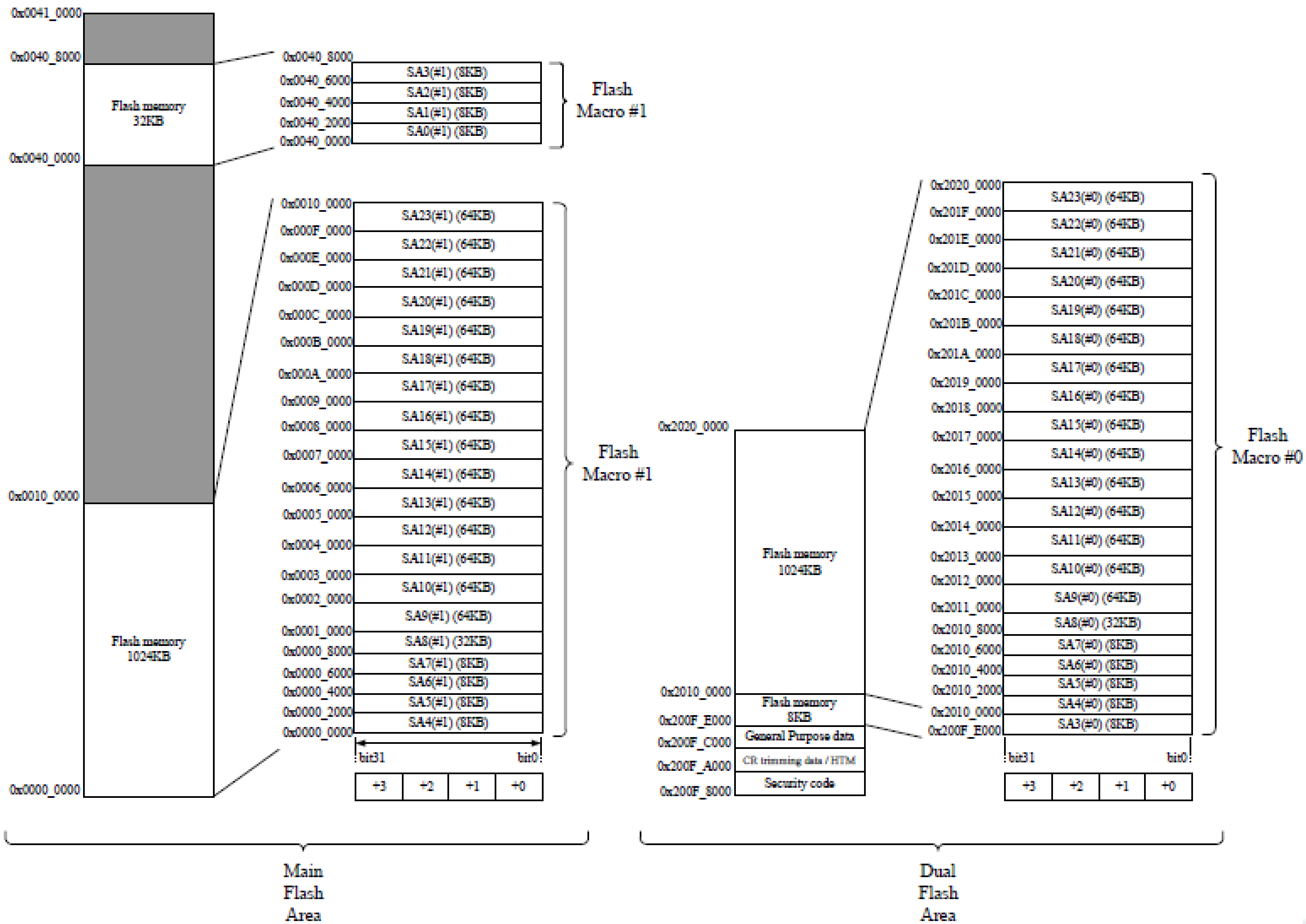
- Different sector sizes
  - Flash erase
  - Sector erase
- Flash security
  - Valid for both banks
- CR trimming



Dual Operation Flash – Dual map mode



Dual Operation Flash – remap mode



More info examples

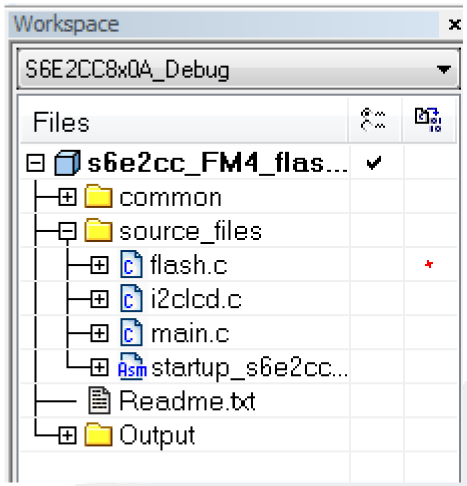
- Even more S6SE2CC examples can be found here:
  - http://www.spansion.com/products/microcontrollers/32-bit-arm-core/fm4/Pages/S6E2CC8L0AGL20000.aspx
- These examples might require minor modifications to run on the SK-S6SE2CC like swaping MFS modules, but porting should be straight forward.
- There will be a similar page in the future that is directly attached to the SK-S6SE2CC.

Dual Flash programming - LAB

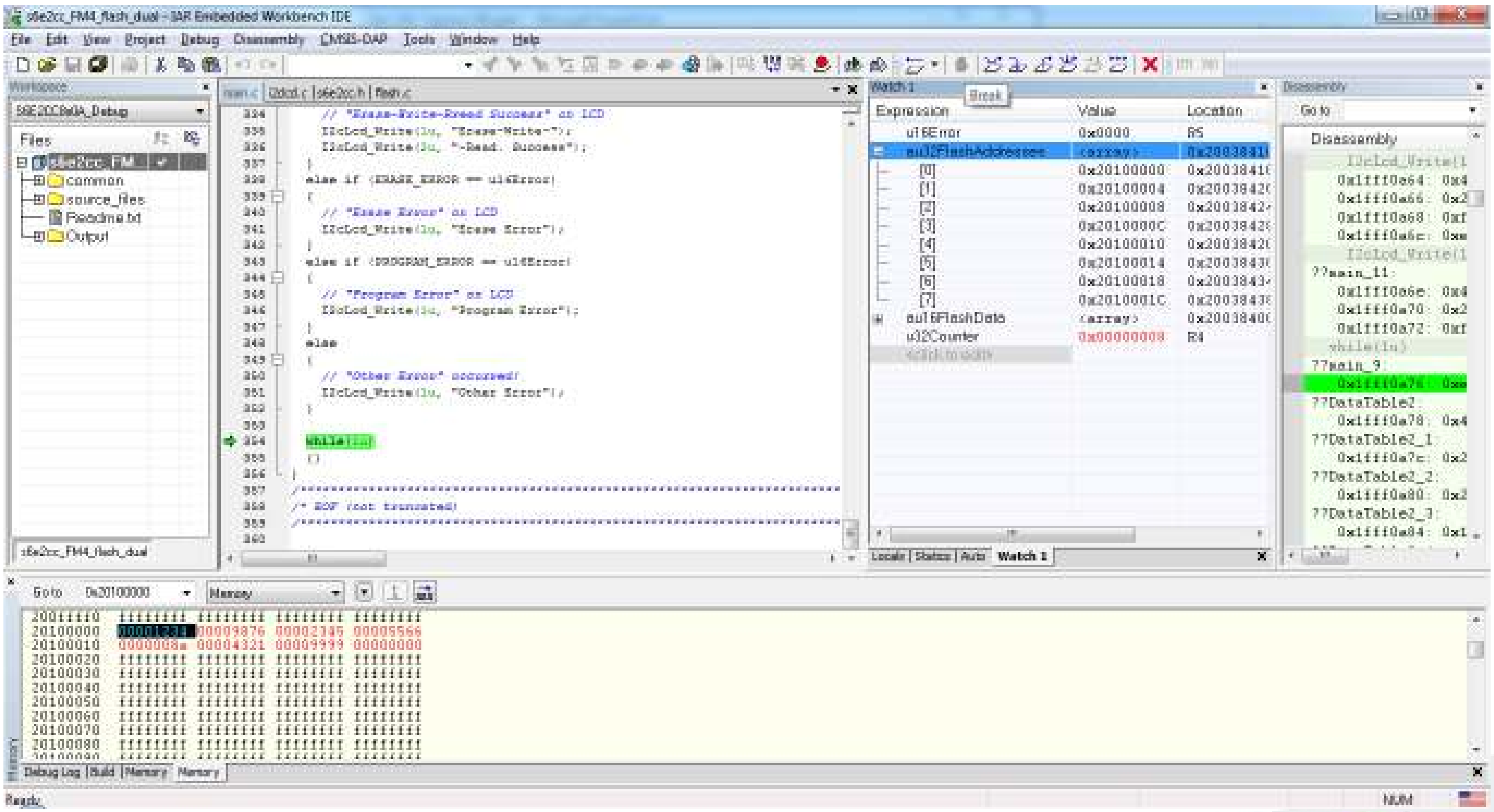
OPEN: Labs\s6e2cc_FM4_flash_dual-v10\example\IAR\s6e2cc_FM4_flash_dual.eww

```
This demo program erases sector SA12 (0x00040000)of the main flash memory while the
flashing code is located within the device's RAM. For further details about main flash,
please refer to the device's flash programming manual.

The erase is roughly checked by reading 0xFFFF from the sector's start address.

The project then writes some data into this sector. Later these data values are
read-back, compared and the result is output on the LCD of
SK-FM4-176-S6SE2CC board:
    Erase-Write-Read Success     : Success
    Erase Error                  : Sector erase failed
    Program Error                : Flash programming failed
```

If you do not have the LCD, check the u16Error value after the run to verify correct execution.



Dual Flash programming - LAB



  CONFIDENTIAL

Dual Flash programming - LAB

- For homework or extra credit – modify the previous labs
  - Make the dual flash programming routine program from one bank to the other without loading the routine into ram.
  - Think out and walk through a failsafe approach to updating the dual bank flash. Consider power brown / black outs, resets, MCU hard fault and watchdog interrupts
  - Analyze the SRAM based code very carefully and determine how much code must really be executed from the ram. Usually just the program command and the while (FALSE == Programmed) loop needs to be in flash. In the past I was able to create a “Do on stack” routine that popped about 10 bytes of code on the stack and that was all at was need to program Flash. Ideas like this can be used to improve overall programming time and aid reliability.

FM4 Flash Memory (Dual Operation Flash)- Lab

Conclusion

- We briefly discussed the internal Flash Module.
- Ran dual mode flash example software demonstrating the ability to program dual banks.
- Ran example of programming internal flash with SRAM based flash loader.
- Don’t forget your homework

www.spansion.com