IIT PALAKKAD

Indian Institute of Technology Palakkad
भारतीय प्रौद्योगिकी संस्थान पालक्काड
Under Ministry of Human Resource Development, Govt. of India
मानव संसाधन विकास मंत्रालय के अधीन, भारत सरकार

Embedded Systems – CS5009 – Lab Session

Embedded Platform:

The embedded platform used for this course is the NXP (formerly Freescale) Freedom Development Platform for Kinetis KL14/15/24/25 MCUs (aka FRDM-KL25Z or KL25Z). The KL25Z provides low cost (less than 1000 INR) platform to explore low power embedded designs.

The FRDM-KL25Z features:

- Kinetis-L MCU (MKL25Z128VLK4)
    - ARM Cortex-M0+ core, up to48MHz CPU speed
    - 128kB FLASH
    - 16kB SRAM
    - DMA
    - UART / 2 SPI / 2 I2C
    - 12-bit DAC
    - 16-bit ADC (up to 24 inputs)
    - USB 2.0 OTG/Host/Device
- Capacitive touch slider
- MMA8451Q accelerometer (I2C)
- Tri-color (RGB) LED
- USB, coin cell battery, external source power supply options
- I/O via Arduino compatible I/O connectors (53 I/O’s available)
- Programmable OpenSDA debug interface

Several software development tool sets support this processor and

specifically, the KL25Z including:

- Codewarrior Development Studio

• IAR Embedded Workbench

• KEIL MDK uVision

• mbed

The KEIL toolset has been selected for this course. A limited free version of is available, MDK-Lite, which is suitable to meet the development needs of this course.

The KEIL MDK uVision toolset features:

• Support for Cortex-M, Cortex-R4, ARM7, and ARM9 devices

• Support for C, C++ and assembly

• µVision4 IDE, debugger, and simulation environment

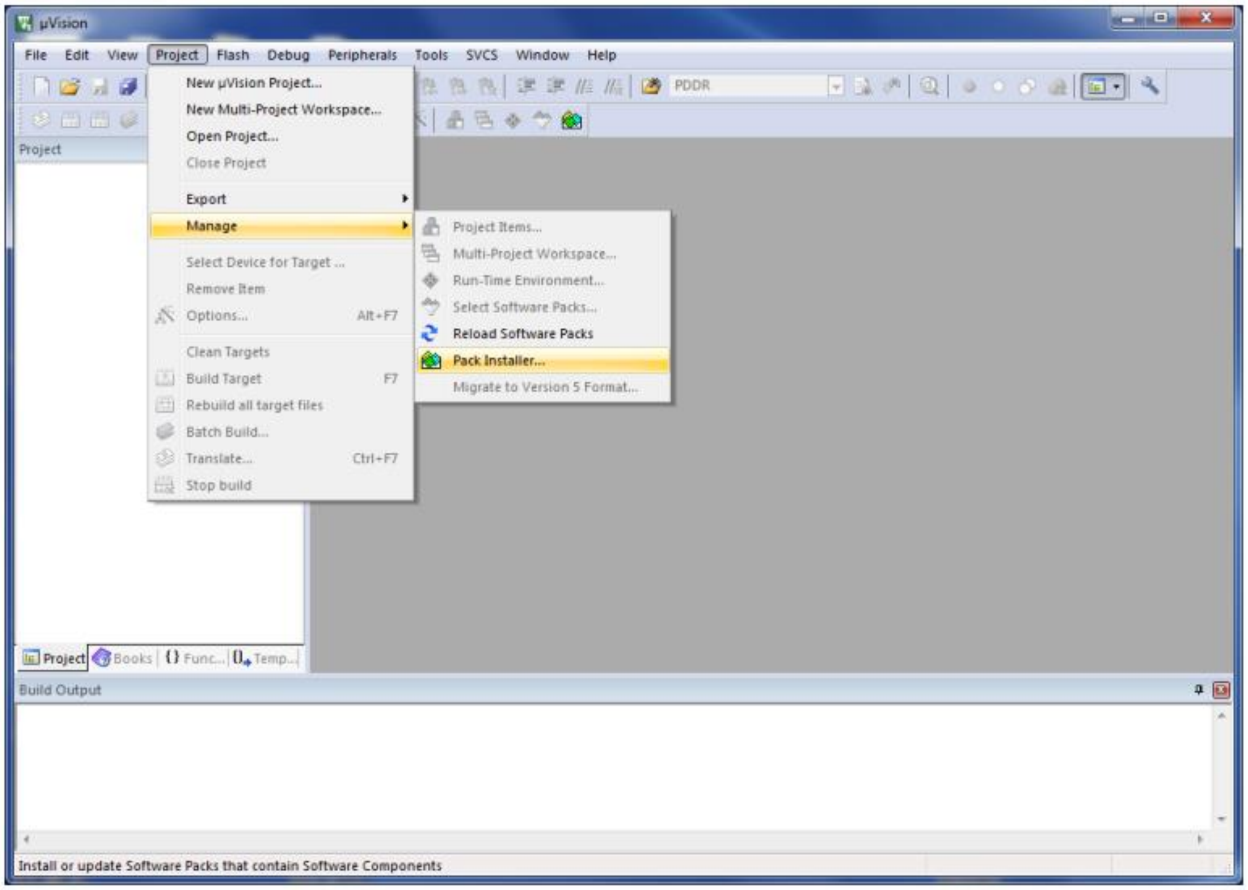
• CMSIS Cortex Microcontroller Software Interface Standard compliant

Tool set-up:

The following steps setup a Windows PC for developing and debugging programs on the FRDM-KL25Z

Step 1: From the KEIL website (registration required), download and install KEIL MDK-ARM uVision 5. Note the free MDK-ARM Lite is sufficient for the lab. (https://www.keil.com/demo/eval/arm.htm)

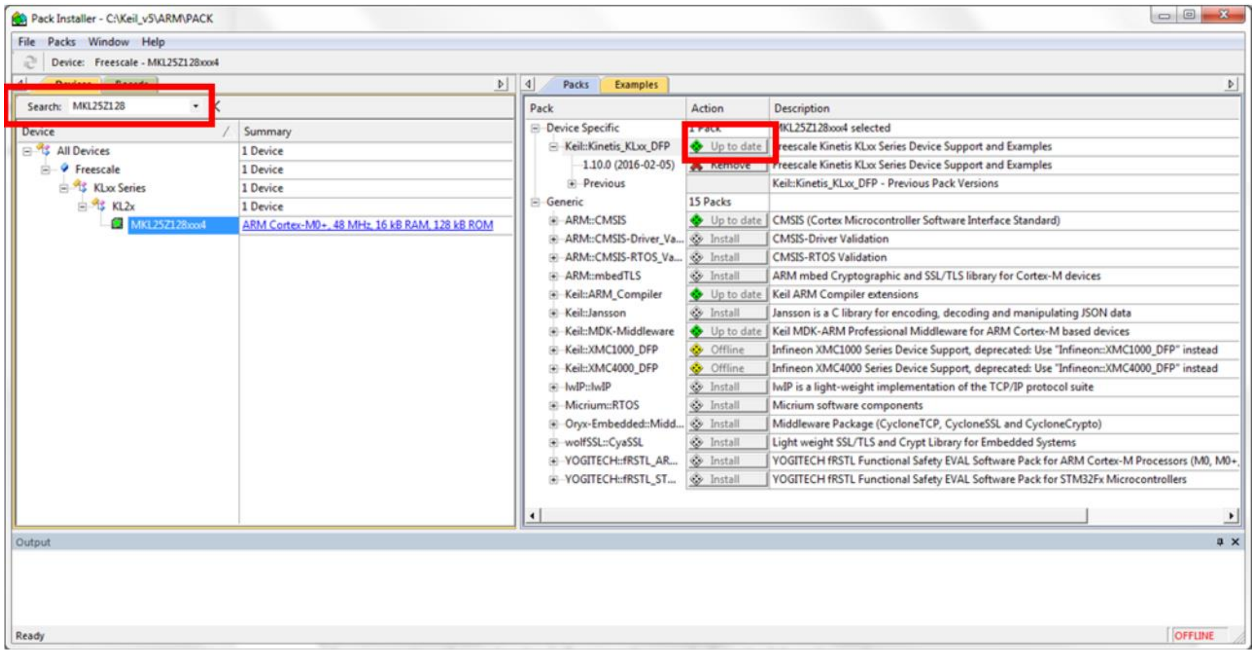
Step 2: Open uVision. Select Project >>> Manage >>> Pack Installer…



Install or update Software Packs that contain Software Components

Step 3: In Pack Installer, search for MKL25Z128 under Devices. Hit

Install on the pack called Keil::Kinetis_KLxx_DFP.



Device	Summary
All Devices	1 Device
Freescale	1 Device
KLxx Series	1 Device
KL2x	1 Device
MKL25Z128xxx4	ARM Cortex-M0+, 48 MHz, 16 kB RAM, 128 kB ROM

Pack	Action	Description
Device Specific	1 Pack	MKL25Z128xxx4 selected
Keil::Kinetis_KLxx_DFP	Up to date	Freescale Kinetis KLxx Series Device Support and Examples
1.10.0 (2016-02-05)	Remove	Freescale Kinetis KLxx Series Device Support and Examples
Previous		Keil::Kinetis_KLxx_DFP - Previous Pack Versions
Generic	15 Packs	
ARM::CMSIS	Up to date	CMSIS (Cortex Microcontroller Software Interface Standard)
ARM::CMSIS-Driver_Va...	Install	CMSIS-Driver Validation
ARM::CMSIS-RTOS_Va...	Install	CMSIS-RTOS Validation
ARM::mbedTLS	Install	ARM mbed Cryptographic and SSL/TLS library for Cortex-M devices
Keil::ARM_Compiler	Up to date	Keil ARM Compiler extensions
Keil::Jansson	Install	Jansson is a C library for encoding, decoding and manipulating JSON data
Keil::MDK-Middleware	Up to date	Keil MDK-ARM Professional Middleware for ARM Cortex-M based devices
Keil::XMC1000_DFP	Offline	Infineon XMC1000 Series Device Support, deprecated: Use "Infineon::XMC1000_DFP" instead
Keil::XMC4000_DFP	Offline	Infineon XMC4000 Series Device Support, deprecated: Use "Infineon::XMC4000_DFP" instead
lwIP::lwIP	Install	lwIP is a light-weight implementation of the TCP/IP protocol suite
Micrium::RTOS	Install	Micrium software components
Oryx-Embedded::Midd...	Install	Middleware Package (CycloneTCP, CycloneSSL and CycloneCrypto)
wolfSSL::CyaSSL	Install	Light weight SSL/TLS and Crypt Library for Embedded Systems
YOGITECH::fRSTL_AR...	Install	YOGITECH fRSTL Functional Safety EVAL Software Pack for ARM Cortex-M Processors (M0, M0+,
YOGITECH::fRSTL_ST...	Install	YOGITECH fRSTL Functional Safety EVAL Software Pack for STM32Fx Microcontrollers

Step 4: From PE Micro website (registration required), download and

install the OpenSDA Windows USB Drivers.

(http://www.pemicro.com/opensda/)

Board setup:

The following steps must be executed to allow code to be loaded and debugged using the KEIL tools and software project used in the labs. This process only needs to be done once initially on a new board or if there are problems connecting to the board. You can check the bootloader and application versions connecting the KL25Z SDA USB connector to a PC and in Windows Explorer, opening the file SDA_INFO.HTM in the drive labeled FRDM-KL25Z. Bootloader version 1.xx and application version 1.xx are the latest at the time this was written. Note that this process may not work on computers running Windows 8 or newer if the KL25Z board has a bootloader older than version 1.11.

Step 1: From PE Micro website (registration required), download and extract the OpenSDA Firmware. (http://www.pemicro.com/opensda/).

Two files will be needed from this file:

1) MSD-DEBUG-FRDM-KL25Z_Pemicro_v118.SDA

2) BOOTUPDATEAPP_Pemicro_vxxx.SDA (which is in a second zip file called OpenSDA_Bootloader_Update_App_vxxx_20xx_xx_xx.zip in the OpenSDA Firmware zip file)

Step 2: Connect the "USB B" end of a "USB B" to "USB Mini" cable to the development PC

Step 3: While holding the RST button on the KL25Z, connect the "USB Mini" connector of the USB cable to the connector labeled SDA on the KL25Z.

Step 4: Release the RST button. The D4 LED should flash green.

Step 5: In Windows Explorer, open the drive labeled BOOTLOADER.

Step 6: Copy the firmware file BOOTUPDATEAPP_Pemicro_v111.SDA to the BOOTLOADER drive.

Step 7: Wait 15 seconds, disconnect the USB cable then reconnect the USB cable, wait another 15 seconds. It is important to wait to ensure the firmware update has time to complete. The D4 LED should flash green when complete.

Step 8: In Windows Explorer, open the drive labeled BOOTLOADER.

Step 9: Copy the firmware file MSD-DEBUG-FRDMKL25Z_Pemicro_v118.SDA to the BOOTLOADER drive.

Step 10: Disconnect and reconnect the USB cable from the KL25Z.

Step 11: In Windows Explorer, the drive should now be labeled FRDMKL25Z. Proper installation can be verified by opening the file SDA_INFO.HTM in the FRDM-KL25Z and verifying the application version matches that of the firmware files that were loaded

In the project create add two files in the source group files **main.c** and **asm_main.s**

Content of the **main.c** file is given below:

```
//#include <MKL25Z4.H>

extern void asm_main(void);

int main (void) {
      asm_main();  // uncomment to use assembly

      while(1){

      }
}
```

The asm_main.s file codes will be updated as the part of this lab progresses.

Part 1:

This step demonstrates the basics of memory access and moving data within the processor. Add the code shown below to the asm_main.s file. The first load moves the 32-bit value (aka word length) from memory at the address const_val to register R1. Then the address assigned to const_val to register R0. Note the '=' loads the address of the value, not the value itself. Then several methods for loading the value assigned to const_val into a register are demonstrated. The first a 32-bit load, then a 16-bit load and finally an 8-bit load. Note how the results are different. The next instruction puts the value associated with the equate equate_val into R0. Note the difference from the constant value move done previously. The next pair of instructions loads the address for const_val into R1. The last two move instructions show how to copy values between register and one way a register can be easily cleared. Variations of these methods are used throughout the course. For example, a very common process is:

Load the address for a Special Function Register (SFR) to a data register (e.g. LDR R0,=SFR_ADDR)

- Load the value to a second register (e.g. LDR R1,=0x12345678)

Code:

```
equate_val EQU 0x8BADF00D
AREA asm_area, CODE, READONLY
EXPORT asm_main
asm_main ; assembly entry point for C function, do not delete
; Add program code here
LDR R1,const_val ;load word (32-bit) from memory
LDR R0,=const_val ;load address to R0
LDR R1,[R0] ;2nd load word (32-bit) from memory
LDRH R1,[R0] ;load half word (16-bit) from mem
LDRB R1,[R0] ;load byte (8-bit) from memory
LDR R0,=equate_val ;load value to R0
LDR R1,=const_val ;load address to R1
MOV R2,R0 ;copy R0 to R2
MOVS R2,#0 ;clear R2
B asm_main
; Put constants here
const_val DCD 0xDEADBEEF
AREA data_area, DATA, READWRITE
; Put variables here
END
```

Part 2:

This step demonstrates some of the basic arithmetic and logic operations. Notice that instructions ending with an ‘S’ modify the application program status register (APSR) with the flags (Z,C,N,V).

```
value1 EQU 50
value2 EQU 123
value3 EQU 0xFFFFFFF0
AREA asm_area, CODE, READONLY
EXPORT asm_main
asm_main ; assembly entry point for C function, do not delete
; Add program code here
MOVS R0,#0 ;clear R0
LDR R1,=value1 ;put value1 in R1
LDR R2,=value2 ;put value2 in R2
LDR R3,=value3 ;put value3 in R3
MSR APSR,R0 ;clear flags
```

```
ADDS R2,R1 ;Add values, update APSR
SUBS R2,R1 ;Subtract values, update APSR
ADDS R3,R1 ;Add values, update APSR
SUBS R3,R1 ;Subtract values, update APSR
MSR APSR,R0 ;clear flags
ADD R3,R1 ;Add values
CMP R1,R2 ;compare
CMP R2,R1 ;compare
CMP R1,R1 ;compare
CMP R1,#0x40 ;compare immediate
CMP R2,#0x40 ;compare immediate
CMP R1,R3 ;compare negative
CMN R1,R3 ;compare negative
B asm_main
; Put constants here
AREA data_area, DATA, READWRITE
; Put variables here
END
```

Part 3:

This step demonstrates program flow control operations using unconditional braches. Modify the code as show below. Each label (i.e. spot1, spot2, spot 3 and spot4) has a memory address associated with the instruction following the label. When the branch instruction (i.e. B spot3) executes occurs, the program counter (R15) is changed to reflect the address associated with the label.

Code:

```
AREA asm_area, CODE, READONLY
EXPORT asm_main
asm_main ; assembly entry point for C function, do not delete
; Add program code here
spot1
B spot3
spot2
B spot4
spot3
B spot2
spot4
```

```
B spot1
; Put constants here
AREA data_area, DATA, READWRITE
; Put variables here
END
```

Part 4:

This step demonstrates the use of conditional branches. Modify the code as show below. Unlike the unconditional branch demonstrated in the previous step, the conditional branch uses the state of the processors flags to control the flow of the program. The branch is only taken if the condition for the specific branch instruction is met. For instance, the BNE (branch not equal) will only branch if the Z flag is cleared. After running the code and recording the results using BNE, rerun the test but replace the BNE with BGE (branch greater than or equal) which branches when N == V.

Code:

```
AREA asm_area, CODE, READONLY
EXPORT asm_main
asm_main ; assembly entry point for C function, do not delete
; Add program code here
rst_cnt
MOVS R0,#3
dec_cnt
SUBS R0,#1
BNE dec_cnt
B rst_cnt
; Put constants here
AREA data_area, DATA, READWRITE
; Put variables here
END
```

Part 5:

This step demonstrates using linked branches for calling subroutines. Modify the code as show below. The branch and link instructions (BL) are uses to call a subroutine. When the BL is executed, the program counter (PC = R15) is changed to reflect the new address and the address for the next instruction after the BL is put into the link register (LR = R14). When the subroutine completes its execution, the branch and exchange (BX LR) instruction copies the link register into the program counter, returning to the instruction after the original function a call.

Code:

```
AREA asm_area, CODE, READONLY
EXPORT asm_main
asm_main ; assembly entry point for C function, do not delete
; Add program code here
loop
LDR R0,=value1 ;call change_val for value1
BL change_value
LDR R0,=value2 ;call change_val for value2
BL change_value
B loop ;do it again
;change_val takes 32-bit value from memory pointed to by R0
;and modifies it by incrementing, then XORing with the
;address, then clearing all byte the lower byte. This is then
;returned back to the address location in memory
change_value
PUSH {R1,R2} ;Save R1 and R2 to stack
LDR R1,[R0] ;Get value from memory
ADDS R1,#1 ;Increment
EORS R1,R0 ;XOR with address
MOVS R2,#0xFF ;Set mask
ANDS R1,R2 ;Mask
STR R1,[R0] ;Save value back to memory
POP {R1,R2} ;Restore R1 and R2
BX LR ;Return
; Put constants here
AREA data_area, DATA, READWRITE
; Put variables here
value1 SPACE 4
value2 SPACE 4
END
```