

# Using DMA to Emulate ADC Flexible Scan Mode with KSDK

Created by Ricardo Olivares Duran on Mar 30, 2015 11:18 AM. Last modified by Ricardo Olivares Duran on Apr 1, 2015 10:01 AM.

This community post describes how to combine the ADC and DMA using KSDK to simulate a flexible peripheral storing the ADC results into a buffer stored in memory. In this configuration the MCU uses less resources, only using one ADC and the hardware (DMA) changing the ADC channel and filling up the memory buffer with the results of the ADC. This way the MCU does not need to read the ADC result register being everything done automatically.

## KSDK

The Kinetis Software Development Kit (SDK) is an extensive suite of robust peripheral drivers, stacks, middleware and example applications designed to simplify and accelerate application development on any Kinetis MCU. The addition of Processor Expert technology for software and board support configuration provides unmatched ease of use and flexibility. The Kinetis SDK is complimentary and includes full source code under a permissive open-source license for all hardware abstraction and peripheral driver software. [KINETIS\\_SDK](#)

## FRDM-K64M

The FRDM-K64F is an ultra-low-cost development platform for Kinetis K64, K63, and K24 MCUs. The FRDM-K64F hardware is form-factor compatible with the Arduino™ R3 pin layout, providing a broad range of expansion board options. Features: MK64FN1M0VLL12 MCU (120 MHz, 1 MB flash memory, 256 KB RAM) for more information see [FRDM-K64F](#)

## DMA to Emulate ADC Flexible Scan

To emulate flexible scan it is necessary to configure the DMA to change the input ADC channel. The channel numbers are defined in the ADC\_mux array. The conversion results are stored in the ADC0\_resultBuffer array with the following order:

1. ADC0\_CH12.
2. ADC0\_CH13.
3. ADC0\_CH23.

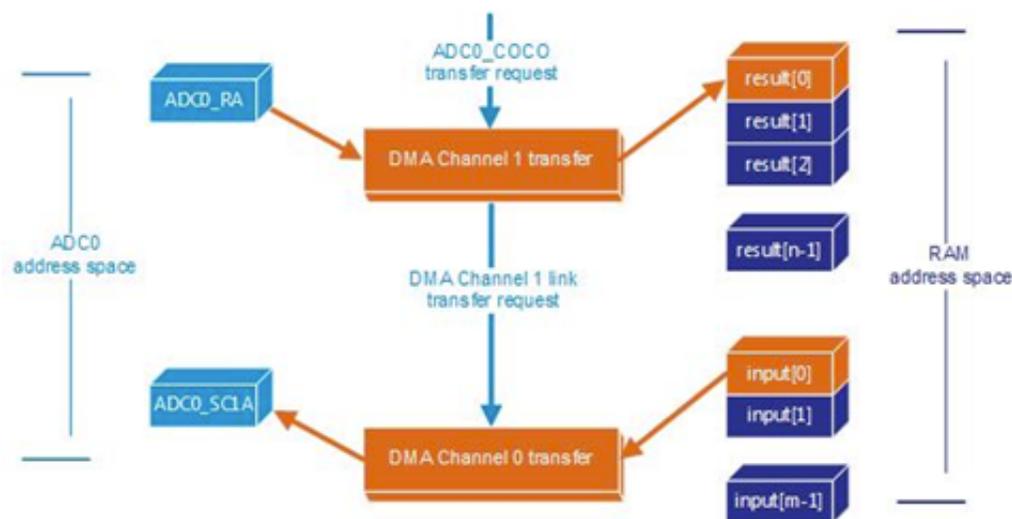
The Kinetis K series of microcontrollers also offers a powerful DMA peripheral with up to 16 channels that can be combined with the ADC to allow the scanning of more than two channels.

## System overview

ADC Flexible Scan mode requires two DMA channels for one ADC converter. DMA channel 1 with a higher priority transfers the resultant ADC data from the ADC0\_RA register to a memory buffer. DMA

channel 0 with a lower priority transfers the next ADC channel setting (input multiplexer channel) from the constant buffer.

The following figure depicts the application



**Figure 1.** Depicts the application

Flow diagram:

The example code that accompanies this application note demonstrates a continuous scan conversion from three ADC channels. Each channel is measured four times, so the result buffer size is  $3 \times 4 = 12$  (the real buffer size is 16, to demonstrate that only 12 data field parts are written). The ADC works in hardware trigger mode, with the LPTMR timer serving as the trigger source. Scanning is executed in continuous mode; thus, after a major loop has finished, the result buffer pointer address\_0 is reloaded and the conversion begins again from the start buffer address. To calculate and change the frequency, check the macro INPUT\_SIGNAL\_FREQ to change the frequency value in Hz. The DMA1 is triggered by the ADC, and the DMA0 is triggered by the eLINK of minor loop of DMA1. In this example the frequency is set to 10 Hz, and the number of samples is 10.

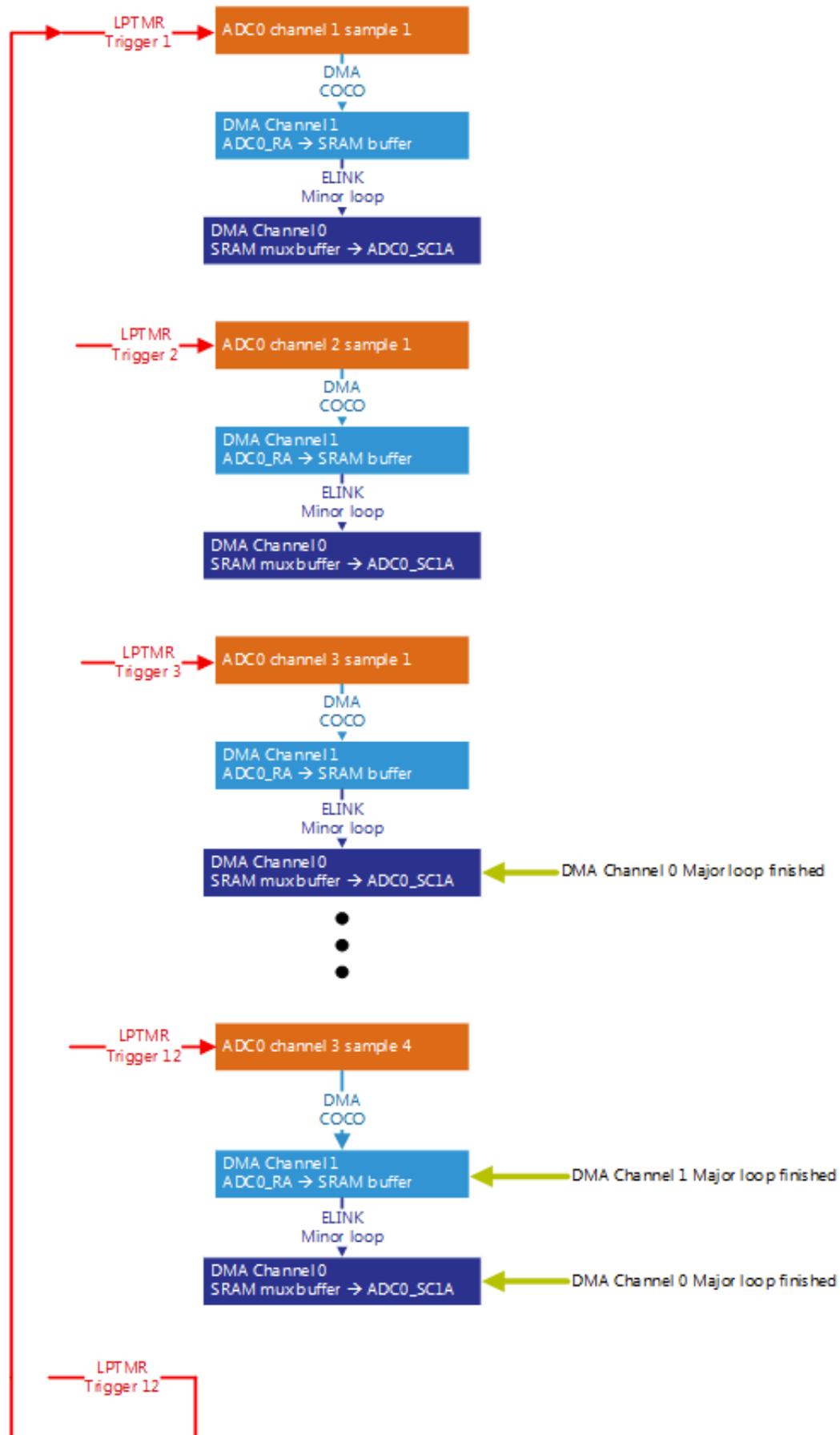


Figure 2. Flow diagram

## Configuration with SDK

Configure the ADC (functions and structures):

The ADC initialization in the application is shown on the following code snippet.

```
The ADC is in interruption mode.  
Trigger by LPTMR (HW trigger)  
    Trigger for eDMA enable  
    Fist channel input is 0x0C. Channel ADC_SE12  
    Select the channel in chnNum
```

```
/*  
 * Initialization ADC for  
 * 12bit resolution, interrrupt mode, hw trigger enabled.  
 * normal convert speed, VREFH/L as reference,  
 * disable continuouse convert mode.  
 */  
ADC_DRV_StructInitUserConfigForIntMode(&adcUserConfig);  
adcUserConfig.hwTriggerEnable = true;  
adcUserConfig.continuousConvEnable = false;  
adcUserConfig.dmaEnable = true;  
  
ADC_DRV_Init(instance, &adcUserConfig, &gAdcState);  
  
/* Install Callback function into ISR. */  
ADC_DRV_InstallCallback(instance, 0U, adc_chn0_isr_callback);  
  
adcChnConfig.chnNum = ADC_INPUT_CHAN;  
adcChnConfig.diffEnable = false;  
adcChnConfig.intEnable = true;  
adcChnConfig.chnMux = kAdcChnMuxOfA;  
  
/* Configure channel0. */  
ADC_DRV_ConfigConvChn(instance, 0U, &adcChnConfig);  
/* Configure channel1, which is used in PDB trigger case. */  
  
return 0;
```

To use the DMA driver, follow these steps:

1. Initialize the DMA module: `EDMA_DRV_Init();`
2. Request a DMA channel: `DMA_DRV_RequestChannel();`
3. Configure the TCD: `EDMA_DRV_PrepareDescriptorTransfer();` and `EDMA_DRV_PushDescriptorToReg();`
4. Register callback function: `EDMA_DRV_InstallCallback();`
5. Start the DMA channel: `EDMA_DRV_StartChannel();`

NOTE: the next two functions are optional for stop and free DMA channel

6. Stop the DMA channel: `EDMA_DRV_StopChannel();`
7. Free the DMA channel: `dma_free_channel();`

Initialize the DMA module: `EDMA_DRV_Init();`

In this function you can select the eDMA channel to use and this function initializes the run-time state structure to provide the eDMA channel allocation release, protect, and track the state for channels. This function also opens the clock to the eDMA modules, resets the eDMA modules and initializes the module to user-defined settings and default settings.

```
chnDMA1.channel= kEDMAChannel1;  
chnDMA0.channel=kEDMAChannel0;
```

```
userConfig.chnArbitration = kEDMAChnArbitrationFixedPriority;  
userConfig2.chnArbitration= kEDMAChnArbitrationRoundrobin;  
userConfig.notHaltOnError = false;  
userConfig2.notHaltOnError = false;
```

```
EDMA_DRV_Init(&state, &userConfig);
```

```
EDMA_DRV_Init(&state2, &userConfig2);
```

Request a DMA channel: `DMA_DRV_RequestChannel()`;

This function allocates eDMA channel according to the required channel allocation and corresponding to the eDMA hardware request, initializes the channel state memory provided by user and fills out the members.

This function provides two ways to allocate an eDMA channel: statically and dynamically. In a static allocation, the user provides the required channel number and eDMA driver tries to allocate the required channel to the user. If the channel is not occupied, the eDMA driver is successfully assigned to the user. If the channel is already occupied, the user gets the return value `kEDMAInvalidChn`, this is request a channel in a static way, In a dynamic allocation, any of the free eDMA channels are available for use. eDMA driver assigns the first free channel to the user.

```
//request ADC1//
uint8_t requestDMA1 = EDMA_DRV_RequestChannel(kEDMAChannel1, kDmaRequestMux0ADC0, &cl
if(kEDMAInvalidChannel==requestDMA1)
    {
        printf("EDMAInvalidChannel 1 . the request is failed.");
    }
//request DMA0//
uint8_t requestDMA0 = EDMA_DRV_RequestChannel(kEDMAChannel0, kDmaRequestMux0AlwaysOnI
if(kEDMAInvalidChannel==requestDMA0)
    {
        printf("EDMAInvalidChannel 0. the request is failed.");
    }
```

DMA configurations: is shown on the following code snippet.

(edma\_transfer\_config\_t) TCD is a configuration structure, inside has a parameters to change for different types to transfers data.

srcAddr: memory address pointing to the source data

destAddr: memory address pointing to the destination address

srcTransferSize: Source data transfer size.

destTransferSize: Destination data transfer size.

srcOffset: Sign-extended offset applied to the current source address form the next-state value as each source read/write is completed.

destOffset: Sign-extended offset applied to the current destination address form the next-state value as each source read/write is completed.

srcLastAddrAdjust: Last source address adjustment.

destLastAddrAdjust: Last destination address adjustment. Note here it is only valid when scatter/gather feature is not enabled.

srcModulo: Source address modulo.

destModulo: Destination address modulo.

minorLoopCount: Minor bytes transfer count. Number of bytes to be transferred in each service request of the channel.

majorLoopCount: Major iteration count.

```

//////////configuration and ELINK DMA channel 1 ////////////
    config[kEDMAChannel1].srcAddr = (uint32_t>(&ADC0_RA); /*!< Memory address po
    config[kEDMAChannel1].destAddr = (uint32_t>(&ADC0_resultBuffer[0]);/*!< Memory
    config[kEDMAChannel1].srcTransferSize = kEDMATransferSize_2Bytes; /*!< Sourc
    config[kEDMAChannel1].destTransferSize = kEDMATransferSize_2Bytes; /*!< Dest
    config[kEDMAChannel1].srcOffset = 0; /*!< Sign-extended offset applied
    form the next-state value as each source
completed. */
    config[kEDMAChannel1].destOffset = 2;
    config[kEDMAChannel1].srcLastAddrAdjust = 0; /*!< Last source address adju
    config[kEDMAChannel1].destLastAddrAdjust = -24; /*!< Last destination addre
valid when scatter/gather feature is not enabled. */
    config[kEDMAChannel1].srcModulo = kEDMAModuloDisable; /*!< Source addre
    config[kEDMAChannel1].destModulo = kEDMAModuloDisable; /*!< Destination
    config[kEDMAChannel1].minorLoopCount = 2; /*!< Minor bytes transfer count.
    in each service request of the channel.
    config[kEDMAChannel1].majorLoopCount = 12; /*!< Major iteration count. */

```

```

stcdDmaChn1.NBYTES.MLNO=0x02;
////////////////////////////////////Elink on////////////////////////////////////LINKCH/////major loop chn1//
stcdDmaChn1.BITER.ELINKNO= (DMA_BITER_ELINKNO_ELINK_MASK|0x0000|0x0C);
stcdDmaChn1.CITER.ELINKNO= (DMA_CITER_ELINKNO_ELINK_MASK|0x0C);
stcdDmaChn1.ATTR= (DMA_ATTR_SSIZE(1)|DMA_ATTR_DSIZE(1));
stcdDmaChn1.CSR=(DMA_CSR_MAJORLINKCH(0)|DMA_CSR_MAJORLINKCH_MASK | DMA_CSR_INTMAJOR_I

```

```

uint16_t statusChnn1 = EDMA_DRV_PrepareDescriptorTransfer(&chnDMA1, &stcdDmaChn1, &c
    if(kStatus_EDMA_Success == statusChnn1)
        {
            statusChnn1 = EDMA_DRV_PushDescriptorToReg(&chnDMA1, &stcdDmaChn1);
        }

```

EDMA\_DRV\_PrepareDescriptorTransfer();

This function sets up the basic transfer for the descriptor.

EDMA\_DRV\_PushDescriptorToReg();

This function copies the software TCD configuration at for the hardware TCD. You needs fill up the structure stcd, and this function do transfer all structure data in DMA registers, is use a especial configuration In this case, the structure is filling up to configuration ELINK mode. The ELINK mode is a configuration of eDMA for triggers other DMA channel in each minion loop transfer is complete.

This is a registers of structure, the mask to enable put on EIINKON mode and for select channel to link and for put on the major loop to finishing trigger channel.

```

stcd.BITER.ELINKNO= (DMA_BITER_ELINKNO_ELINK_MASK|0x0000|0x0C);
stcd.CITER.ELINKNO= (DMA_CITER_ELINKNO_ELINK_MASK|0x0C);

```

DMA\_BITER\_ELINKNO\_ELINK\_MASK it's a mask to active the channel-to-channel linking on minor-loop complete. As the channel completes the minor loop, this flag enables linking to another channel, defined by the LINKCH field. The link target channel initiates a channel service request via an internal mechanism

You can see more information in Reference manual.

```

////////configuration DMA channel0 ////////////////////////////////////////////
    config[kEDMAChannel0].srcAddr = (uint32_t>(&ADC_mux[0]));    /*!< Memory address
        config[kEDMAChannel0].destAddr = (uint32_t>(&ADC0_SC1A));    /*!< Memory address:
        config[kEDMAChannel0].srcTransferSize = kEDMATransferSize_1Bytes;    /*!< Sourc
        config[kEDMAChannel0].destTransferSize = kEDMATransferSize_1Bytes;    /*!< Dest:
        config[kEDMAChannel0].srcOffset = 1;    /*!< Sign-extended offset applica
                                form the next-state value as each source i
completed. */
        config[kEDMAChannel0].destOffset = 0;
        config[kEDMAChannel0].srcLastAddrAdjust = -3;    /*!< Last source address adj
        config[kEDMAChannel0].destLastAddrAdjust = 0;    /*!< Last destination address
valid when scatter/gather feature is not enabled. */
        config[kEDMAChannel0].srcModulo = kEDMAModuloDisable;    /*!< Source address:
        config[kEDMAChannel0].destModulo = kEDMAModuloDisable;    /*!< Destination
        config[kEDMAChannel0].minorLoopCount = 1;    /*!< Minor bytes transfer count.
                                in each service request of the channel.
        config[kEDMAChannel0].majorLoopCount = 3;    /*!< Major iteration count. */

```

```

uint16_t statusChnn0 = EDMA_DRV_PrepareDescriptorTransfer(&chnDMA0, &stcdDmaChn0,
    if(kStatus_EDMA_Success == statusChnn0)
    {
        statusChnn0 = EDMA_DRV_PushDescriptorToReg(&chnDMA0, &stcdDmaChn0);
    }

```

The function drivers in main file:

```
hardware_init();
dbg_uart_init();
OSA_Init();
init_adc(ADC_INST);
init_trigger_source(ADC_INST);
config_DMA();
```

Open your terminal (baud rate 115200)

The results will appear on the terminal software.

```
ADC_eDMA demo running...
results ADC=   chn13, chn12, chn23,  chn13, chn12, chn23, chn13, chn12, chn23, chn13, chn12, chn23
DMA1 RAMbuffer= 0fff, 0002, 05b8, 0fff, 0001, 059d, 0fff, 0001, 05a6, 0fff, 0001, 05aa,
DMA1 RAMbuffer= 0fff, 0002, 05aa, 0fff, 0001, 05ac, 0fff, 0001, 05ad, 0fff, 0001, 05a3,
DMA1 RAMbuffer= 0fff, 0001, 05ab, 0fff, 0001, 05af, 0fff, 0001, 05b3, 0fff, 0001, 05af,
DMA1 RAMbuffer= 0fff, 0008, 05b9, 0fff, 0001, 05ad, 0fff, 0001, 05aa, 0fff, 0001, 05b2,
DMA1 RAMbuffer= 0ffe, 0001, 05ab, 0fff, 0001, 05b2, 0fff, 0002, 05b3, 0fff, 0001, 05b7,
DMA1 RAMbuffer= 0fff, 0001, 05ba, 0fff, 0001, 05ae, 0fff, 0001, 05bf, 0fff, 0001, 05b0,
█
```

**Figure 3.** Screen serial

You can also see the results in the debugger window.

Expression	Type	Value
ADC0_resultBuffer	uint16_t [16]	0x20000000
(x)- ADC0_resultBuffer[0]	uint16_t	0xff
(x)- ADC0_resultBuffer[1]	uint16_t	0x1
(x)- ADC0_resultBuffer[2]	uint16_t	0x5ba
(x)- ADC0_resultBuffer[3]	uint16_t	0xff
(x)- ADC0_resultBuffer[4]	uint16_t	0x1
(x)- ADC0_resultBuffer[5]	uint16_t	0x5ae
(x)- ADC0_resultBuffer[6]	uint16_t	0xff
(x)- ADC0_resultBuffer[7]	uint16_t	0x1
(x)- ADC0_resultBuffer[8]	uint16_t	0x5bf
(x)- ADC0_resultBuffer[9]	uint16_t	0xff
(x)- ADC0_resultBuffer[10]	uint16_t	0x1
(x)- ADC0_resultBuffer[11]	uint16_t	0x5b0

**Figure 4.** result buffer

### Steps to include ADC Flexible Scan software to KSDK

In order to include this demo in the KSDK structure, the files need to be copied into the correct place. The `adc_dma_demo` folder should be copied into the `<KSDK_install_dir>/demos` folder.

If the folder is copied to a wrong location, paths in the project and makefiles will be broken. When the copy is complete you should have the following locations as paths in your system:

- `<KSDK_install_dir>/demos/ adc_dma_demo /iar`
- `<KSDK_install_dir>/demos/ adc_dma_demo /kds`
- `<KSDK_install_dir>/demos/ adc_dma_demo /src`

In addition, to build and run the demo, it is necessary to download one of the supported Integrated Development Environment (IDE) by the demo:

- [Freescale Kinetis Design Studio \(KDS\)](#)
- [IAR Embedded Workbench](#)

Once the project is opened in one of the supported IDEs, remember to build the KSDK library before building the project, if it was located at the right place no errors should appear, start a debug session and run the demo.

[adc\\_dma\\_demo.zip](#) ⓘ

210.4 K

585 Views

Categories:

Tags:

Average User Rating

(0 ratings)

## 2 Comments

amin iman Oct 26, 2015 6:27 AM

hi ricardo,

i find some problem. there is "Error retrieving content description for resource '/adc\_dma\_demo\_frdmk64f120m/board/board.h'." its not only board.h, but other .h in board folder cant detect. i was located the folder of source code in "C:\Freescale\KSDK\_1.2.0\examples\frdmk64f\demo\_apps\adc\_dma\_demo" and making the path as you can see in attach below.

hope you can help me to solve the problem. furthermore, many thanks for you if you can make step by step (screenshot project) integrated to KDS especially.

regards,  
amin

Actions

 Like (0)

amin iman Oct 26, 2015 8:16 AM (in response to amin iman)

sorry lose of the attachment



Actions 👍 Like (0)

[Freescale Worldwide](#) | [Media](#) | [Investors](#) | [Partners](#) | [University Programs](#) | [Events](#) | [Careers](#)  
[Terms of Use](#) | [Privacy](#) | [Cookies](#) | [Terms of Sale](#) | [Newsletter](#) | [Contact Us](#) | [Mobile Apps](#) | [Mobile Site](#)  
[Social @Freescale](#) | [Blogs](#) | [Follow Us](#)

[Home](#) | [Top of page](#) | [Help](#)

© 2015 Jive Software | Powered by **jive**