**K64 SPI Slave to beaglebone Linux Master project**

After installing Kinetis Design Studio on Windows.

Steps to install and work with the FRDM k64 with Kinetis SDk.

<https://github.com/EliArad/MyDocuments/blob/master/Kinetis_K64/First%20steps%20with%20Kinetis%20SDk.docx>

The starting example source code:

SDK\_2.1\_FRDM-K64F-AGM01\boards\frdmk64f\rtos\_examples\freertos\_dspi\kds

The DSPI example contains both Master and slave code.

We need only the slave code, we can remove the master code.

**Pin mux and connection**

The example is using SPI1 for the Slave, but SPI1 should be solder and does not have connector

**#define** EXAMPLE\_DSPI\_SLAVE\_BASE (SPI1\_BASE)

SPI 1 can be in several places, the pin mux that was selected

PORT\_SetPinMux(PORTD, PIN4\_IDX, *kPORT\_MuxAlt7*); /\* PORTD4 (pin A4) is configured as SPI1\_PCS0 \*/

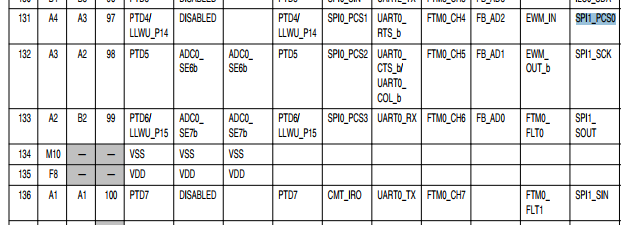
PORT\_SetPinMux(PORTD, PIN5\_IDX, *kPORT\_MuxAlt7*); /\* PORTD5 (pin A3) is configured as SPI1\_SCK \*/

PORT\_SetPinMux(PORTD, PIN6\_IDX, *kPORT\_MuxAlt7*); /\* PORTD6 (pin A2) is configured as SPI1\_SOUT \*/

PORT\_SetPinMux(PORTD, PIN7\_IDX, *kPORT\_MuxAlt7*); /\* PORTD7 (pin A1) is configured as SPI1\_SIN \*/

Will probably be located in the external header connector.

<http://www.nxp.com/docs/en/data-sheet/K64P144M120SF5.pdf> page 74



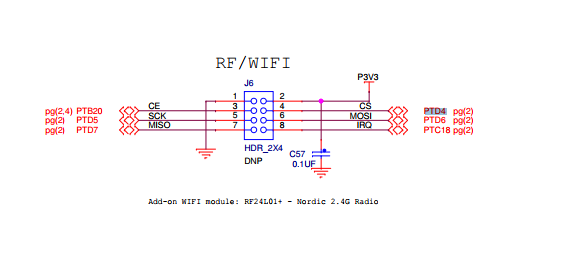
It is true, alternate 7

Now we just need to find those ports in the schematic to connect the beaglebone black.

<http://www.nxp.com/downloads/en/schematics/FRDM-K64F-SCH-E4.pdf>

SP2 are not exported fully in the FRDM connector header.

SPI1 is not a header place holder, we need to solder it.



Connector header: J6

|  |  |  |
| --- | --- | --- |
| Port and pin | Signal name | Pin on Connector |
| PTD4 | Chip select | J6 pin 4 |
| PTD5 | Clock | J6 pin 5 |
| PTD6 | Spi Out MOSI | J6 pin 6 |
| PTD7 | Spi In MISO | J6 pin 7 |
|  | Ground | J6 pin 1 |

The MISO should connect to MISO and MOSI to MOSI , ground to ground, CS to CS

But connector J6 does not good for inserting wires, only soldering

**We shell use SPI0**

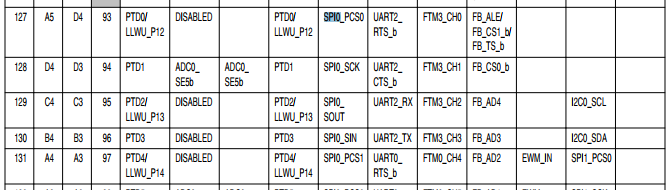
PORT\_SetPinMux(PORTD, PIN0\_IDX, *kPORT\_MuxAlt2*); /\* PORTD0 (pin A5) is configured as SPI0\_PCS0 \*/

PORT\_SetPinMux(PORTD, PIN1\_IDX, *kPORT\_MuxAlt2*); /\* PORTD1 (pin D4) is configured as SPI0\_SCK \*/

PORT\_SetPinMux(PORTD, PIN2\_IDX, *kPORT\_MuxAlt2*); /\* PORTD2 (pin C4) is configured as SPI0\_SOUT \*/

PORT\_SetPinMux(PORTD, PIN3\_IDX, *kPORT\_MuxAlt2*); /\* PORTD3 (pin B4) is configured as SPI0\_SIN \*/

|  |  |  |
| --- | --- | --- |
| Port and pin | Signal name | Pin on Connector |
| PTD0 | Chip select | J2 pin 6 |
| PTD1 | Clock | J2 pin 12 |
| PTD2 | Spi Out MOSI | J2 pin 8 |
| PTD3 | Spi In MISO | J2 pin 10 |
|  | Ground | J2 pin 14 |



Code overview:

**static** **void** **slave\_task**(**void** \*pvParameters)

Is the RTOs task entry.

As for start we can try use the DSP example.

I prefer to reduce the code there.

The interrupt of the slave was reduce to minimum

**void** **DSPI\_SlaveTransferHandleIRQ**(SPI\_Type \*base, dspi\_slave\_handle\_t \*handle)

{

assert(handle);

uint8\_t dummyPattern = DSPI\_DUMMY\_DATA;

uint32\_t dataReceived;

uint32\_t dataSend = 0;

**while** (DSPI\_GetStatusFlags(base) & *kDSPI\_RxFifoDrainRequestFlag*)

{

dataReceived = base->POPR;

PRINTF("dr = %d\n" , dataReceived);

base->PUSHR\_SLAVE = dataSend;

DSPI\_ClearStatusFlags(base, *kDSPI\_RxFifoDrainRequestFlag*);

}

**return**;

}

The idea is to hold a fifo here for input and output and serve the code from here.

Simple and faster like in I2C.

From the DSPI example , I throw away all the master code and change the slave to use SPI0

**static** **void** **slave\_task**(**void** \*pvParameters)

{

dspi\_slave\_config\_t slaveConfig;

dspi\_transfer\_t slaveXfer;

uint32\_t i;

callback\_message\_t cb\_msg;

cb\_msg.sem = xSemaphoreCreateBinary();

dspi\_sem = cb\_msg.sem;

**if** (cb\_msg.sem == NULL)

{

PRINTF("DSPI slave: Error creating semaphore\r\n");

vTaskSuspend(NULL);

}

**for** (i = 0; i < TRANSFER\_SIZE; i++)

{

slaveSendBuffer[i] = 9;

slaveReceiveBuffer[i] = 7;

}

slaveConfig.whichCtar = *kDSPI\_Ctar0*;

slaveConfig.ctarConfig.bitsPerFrame = 8;

slaveConfig.ctarConfig.cpol = *kDSPI\_ClockPolarityActiveHigh*;

slaveConfig.ctarConfig.cpha = *kDSPI\_ClockPhaseFirstEdge*;

slaveConfig.enableContinuousSCK = false;

slaveConfig.enableRxFifoOverWrite = false;

slaveConfig.enableModifiedTimingFormat = false;

slaveConfig.samplePoint = *kDSPI\_SckToSin0Clock*;

/\* Set dspi slave interrupt priority higher. \*/

NVIC\_SetPriority(EXAMPLE\_DSPI\_SLAVE\_IRQN, DSPI\_NVIC\_PRIO);

DSPI\_SlaveInit(EXAMPLE\_DSPI\_SLAVE\_BASEADDR, &slaveConfig);

/\*Set up slave first \*/

DSPI\_SlaveTransferCreateHandle(EXAMPLE\_DSPI\_SLAVE\_BASEADDR, &g\_s\_handle, DSPI\_SlaveUserCallback, &cb\_msg);

/\*Set slave transfer ready to receive/send data\*/

slaveXfer.txData = slaveSendBuffer;

slaveXfer.rxData = slaveReceiveBuffer;

slaveXfer.dataSize = TRANSFER\_SIZE;

slaveXfer.configFlags = *kDSPI\_SlaveCtar0*;

DSPI\_SlaveTransferNonBlocking(EXAMPLE\_DSPI\_SLAVE\_BASEADDR, &g\_s\_handle, &slaveXfer);

/\* Wait for transfer to finish \*/

xSemaphoreTake(cb\_msg.sem, portMAX\_DELAY);

**for** (**int** i = 0 ; i < 16 ; i++)

{

PRINTF("slaveXfer.rxData = %x\n" , slaveXfer.rxData[i]);

}

slaveXfer.txData = slaveSendBuffer;

slaveXfer.rxData = slaveReceiveBuffer;

slaveXfer.dataSize = TRANSFER\_SIZE;

//errorCount = 0;

vTaskSuspend(NULL);

}