Embedding it better...

µTasker Document

- **Kinetis demo user's guide**

Table of Contents

Introduction

The µTasker project has been extended to include support of the Freescale Kinetis family of devices.

This guide explains how to start using the project together with these devices based on the TWR-K60N512 board either stand-alone or together with the Tower kit, including USB, Ethernet and TFT display with touch screen.

The project includes IAR6, Rowley Crossworks 2.1, Keil uVision 4.14 and Codewarrior v10.2 (based on Eclipse) configurations as well as a stand-alone GCC make file build. In addition it allows the Kinetis and external peripherals such as the TFT display to be simulated in the µTasker using VisualStudio.

Appendix A contains developer's notes concerning the Kinetis devices and also details about the operation of the µTasker projects hardware related code for these parts.

1. Preparing to start

The minimum HW requirements for starting with the µTasker project on the Kinetis K60N512 is the TWR-K60N512 module for the tower kit. This module can be powered via USB and includes an on-board OSJTAG debugger which allows loading code and debugging with either IAR6, Rowley Crossworks, uVision or Codewarrior v10.2.

When no board is available, but also generally for high-speed project development, the µTasker simulator (based on Microsoft VisualStudio) can be used to simulate the processor and its peripherals for improved debugging, code development and verification efficiency.

If the complete Tower Kit is available, the TWR-K60N512 module can be used in combination with this in order to work with extended peripherals.

The first steps thus involve installing the software:

- VisualStudio C++ [µTasker simulator IDE]
- IAR6 [IDE]
- Codewarrior v10.2 [IDE]
- Rowely Crossworks for ARM V2.1 [IDE]
- Keil uVision4 for ARM V4.14 [IDE]
- P & E Kinetis Tower Toolkit [various utilities and OSJTAG drivers]

The IDEs are optional and so not all need to be installed. The P & E Kinetis Tower Toolkit should be installed (included on the CD delivered with the TWR-K60N512 board) to ensure that the debugger drivers are available.

Once the software installation has completed (a system restart may also be required) the TWR-K60N512 can be connected to the development PC via its USB device adapter (J13) and will be recognised as OSBDM debug port. Any pre-installed software on the board will now also run, allowing verification that the hardware is operational.

Since the K60N512 has 128k of SRAM it is possible to execute fairly interesting code directly from RAM rather than having to program the FLASH. As soon as the program size and its use of SRAM exceeds this amount, the project needs to be loaded to FLASH and executed from there.

2. Compiling the µTasker Demo using CodeWarrior v10.2

The µTasker software can be obtained on request from www.uTasker.com/Licensing/request.html.

The project is free of charge for all non-commercial use and can be evaluated for a period of 30 days for commercial purposes. Full licensing details can be read at www.uTasker.com/Licensing/License.html

Precompiled target code is freely available of the software resource page: www.uTasker.com/software/software.html which can loaded to the board without the need to compile it.

The first time that Codewarrior is started it requires a workspace to be defined. The workspace can be located anywhere and the project will later be imported into the workspace.

Once Codewarrior has started, the µTasker project can be imported. This is achieved by selecting the context menu (right mouse click) in the "CodeWarrior Projects" window and commanding "Import…". In the import dialog the action "General | Existing Projects into Workspace" can be selected and then the root directory set to uTaskerV1.4 project directory. The import doesn't need to copy the project to the workspace.

Once the import has been executed the project is seen as uTaskerV1.4 in the CodeWarrior Project window, with two targets "MK60N512VM100_INTERNAL_FLASH" and "MK60N512VM100_INTERNAL_RAM".

Ensure that the target board is configured for the TWR-K60N512 board in the µTasker V1.4 project file `config.h` as follows:

`#define TWR_K60N512` must be active to select the Tower board.

The project is compiled by clicking on the build button (Ctr + B will start a complete build).

The target code size is about 89k when Ethernet [including FTP, Web server, Telnet and uFileSystem], USB and TFT are active.

Smaller projects can be loaded and executed from SRAM but larger projects will need to be loaded to FLASH (MK60N512VM100_INTERNAL_FLASH configuration), which can be performed by executing the debug command (configure as default to load to FLASH using the OSJTAG interface).

CodeWarrior also supports loading ELF or SREC files with its Flash Programmer utility.

The project is configured to place output data (elf, s19, map file etc.) in the project directories `\uTaskerV1.4\Applications\uTaskerV1.4\KinetisCodeWarrior\MK60N512VMD100_INTERNAL_FLASH` or `\uTaskerV1.4\Applications\uTaskerV1.4\KinetisCodeWarrior\MK60N512VMD100_INTERNAL_RAM`.

Note that minor hardware modifications may be required to run the TFT demo – see the section about the TFT demo for more details.

3. Compiling the µTasker Demo using IAR6

The µTasker software can be obtained on request from www.uTasker.com/Licensing/request.html.

The project is free of charge for all non-commercial use and can be evaluated for a period of 30 days for commercial purposes. Full licensing details can be read at www.uTasker.com/Licensing/License.html

Precompiled target code is freely available on the software resource page: www.uTasker.com/software/software.html which can loaded to the board without the need to compile it.

Open the project `\Applications\uTaskerV1.4\IAR_Kinetis\uTaskerV1.4.ewp` in IAR Embedded Workbench. Select the target `Debug` to run from SRAM or `Release` to run from FLASH and ensure that the target board is configured for the TWR-K60N512 board in the µTasker V1.4 project file `config.h` as follows:

`#define TWR_K60N512` must be active to select the Tower board.

Compile the project by pressing F7 (`Project | make`) or by clicking on the “Make” icon. The project should build with no errors but there may be some warnings which can be ignored. The target code size is about 78k when Ethernet [including FTP, Web server, Telnet and uFileSystem], USB and TFT are active.

In case the kick-start version is used (limited to 32k) either Ethernet can be deactivated by disabling ETH_INTERFACE in config.h or the TFT display can be deactivated by disabling SUPPORT_GLCD in config.h.

The debugger “PE Micro” should be selected and the code can be loaded to the board via the OSJTAG connection by clicking on the “Download and Debug” icon (with green arrow). Depending on the target selected this will be loaded either to SRAM or programmed to FLASH.

Note that minor hardware modifications may be required to run the TFT demo – see the section about the TFT demo for more details.

4. Compiling the µTasker Demo using Rowely Crossworks 2.1

The µTasker software can be obtained on request from www.uTasker.com/Licensing/request.html.

The project is free of charge for all non-commercial use and can be evaluated for a period of 30 days for commercial purposes. Full licensing details can be read at www.uTasker.com/Licensing/License.html

Precompiled target code is freely available of the software resource page: www.uTasker.com/software/software.html which can loaded to the board without the need to compile it.

Open the project `\Applications\uTaskerV1.4\Rowley_Kinetis\uTaskerV1_4.hzp` in Crossworks for ARM V2.1. Ensure that the target board is configured for the TWR-K60N512 board in the µTasker V1.4 project file `config.h` as follows:

`#define TWR_K60N512` must be active to select the Tower board.

Compile the project by pressing F7 (`Build | Build uTaskerV14`) or by clicking on the "Build" icon. The project should build with neither errors but there may be some warnings which can be ignored. The target code size is about 82k when Ethernet [including FTP, Web server, Telnet and uFileSystem], USB and TFT are active.

The debugger "Kinetis OSJTAG" should be selected and the code can be loaded to the board via the OSJTAG connection by clicking on the "Start Execution" icon (with blue arrow) – (F5)

Note that minor hardware modifications may be required to run the TFT demo – see the section about the TFT demo for more details.

5. Compiling the µTasker Demo using Keil uVision 4.14

The µTasker software can be obtained on request from www.uTasker.com/Licensing/request.html.

The project is free of charge for all non-commercial use and can be evaluated for a period of 30 days for commercial purposes. Full licensing details can be read at www.uTasker.com/Licensing/License.html

Precompiled target code is freely available of the software resource page: www.uTasker.com/software/software.html which can loaded to the board without the need to compile it.

Open the project `\Applications\uTaskerV1.4\uVision_Kinetis\uTaskerV1.4.uvproj` in Keil uVision4.

Ensure that the target board is configured for the TWR-K60N512 board in the µTasker V1.4 project file `config.h` as follows:

`#define TWR_K60N512` must be active to select the Tower board.

Compile the project by pressing F7 (`Project | Build target`) or by clicking on the "Build" icon. The project should build with no errors but there may be some warnings which can be ignored. The target code size is about 52k when Ethernet [including FTP, Web server, Telnet and uFileSystem], USB and TFT are active.

In case the evaluation version is used (limited to 32k) either Ethernet can be deactivated by disabling `ETH_INTERFACE` in `config.h` or the TFT display can be deactivated by disabling `SUPPORT_GLCD` in `config.h`.

The present version of uVision4 doesn't support the OSJTAG interface and so requires a JTAG debugger to be connected (warning – the TWR-K60N512 has a fine-pitch 20 pin CORTEX connector which may require an additional adapter to connect to standard debug cables).

Note that minor hardware modifications may be required to run the TFT demo – see the section about the TFT demo for more details.

6. Compiling the Project in VisualStudio C++ and with GCC

The µTasker simulator operates with VisualStudio C++ from version 6.0 and upwards, including the free VisualStudio Express editions. This operation is detailed in the document http://www.utasker.com/docs/KINETIS/uTaskerV1.4-Tutorial-Kinetis.PDF

The VisualStudio environment includes also a very powerful editor and the capabilities to add post-build steps which are suitable for cross-compiling to the target hardware. As detailed in the tutorial in the link above, a target is included which will automatically perform this stage to generate a standalone binary image using the GNU compiler, plus images suitable for downloading to the target with boot loader support.

As long as the GNU compiler is installed on the PC, the project can be comfortably built from the VisualStudio environment with GCC. In this case the target configuration "`Win32 uTasker kinetis plus GNU build`" should be selected.

7. Recommended Jumper Settings on the TWR-K60N512, TWR-K60F120M and K70F120M

The TWR-K60N512 board can be used standalone but its functions are only limited since it doesn't have many peripherals that are readily accessible and it also doesn't have any on-board Ethernet physical layer device or socket. When running as standalone board the CPU is supplied with a 50MHz clock signal from a local oscillator Y1.

If the board is inserted into the tower kit and is to use the Ethernet connection on the serial board (TWR-SER) it should use a 50MHz signal supplied by the serial board via the back plane as clock. This is the 50MHz signal that is used by the PHY on the serial board, which should be synchronised to the CPU clock for proper Ethernet operation. (*See the appendix section on Ethernet for more details*).

The K60 Ethernet connection is RMII (reduced media-independent interface) and the TWR-SER board must also be correctly configured in this mode for Ethernet operation. These two configurations are shown in the following jumper table.

	TWR-K60N512/ TWR-K53N512	**TWR-K60F120M/ TWR-K70F120M**	**TWR-SER**
Stand-alone operation	J6 1-2 (K60) J17 ON (K53) Local 50MHz oscillator input	J18 ON (K60) or J19 ON (K70) Local 50MHz oscillator input	Not applicable
Ethernet RMII operation	J6 2-3 (K60) J17 OFF (K53) PHY Clock via backplane	J18 OFF (K60) or J19 OFF (K70) PHY Clock via backplane	J2 3-4 selects 50MHz clock for the PHY and the backplane J3 2-3 Enables 50MHz clock to drive backplane line CLOCKIN0 J12 9-10 Sets a pull-up on the PHY's CONFIG0 strap input to configure it to RMII mode J6 7-8 IRQ1 interrupt line for PHY interrupt*
Ethernet MII operation	This is not possible with the TWR-K60N512/TWR-K53N512 but is used with the Kirin3 processor board	This is not possible with the TWR-K60F120M/ TWR-K70F120M but is used with the Kirin3 processor board	J2 1-2 selects 25MHz clock for the PHY (MII mode) J3 2-3 – none connected J12 9-10 – none connected J6 7-8 IRQ1 interrupt line for PHY interrupt

*The interrupt line from the PHY on the TWR-SER card can be connected to several backplane lines. **J6 7-8** should be selected on the TWR-SER board so that the IRQ1 line is selected. This is connected to port PA.27 on the K60 on the TWR-K60N512 board and PB.07 on the TWR-K53N512, TWR-K60F120M and TWR-K70F120M boards.

8. uGLCDLIB TFT demo

In order to see the TFT in operation the project define `SUPPORT_GLCD` can be set. The TFT is selected by enabling `TFT2N0369_GLCD_MODE`.

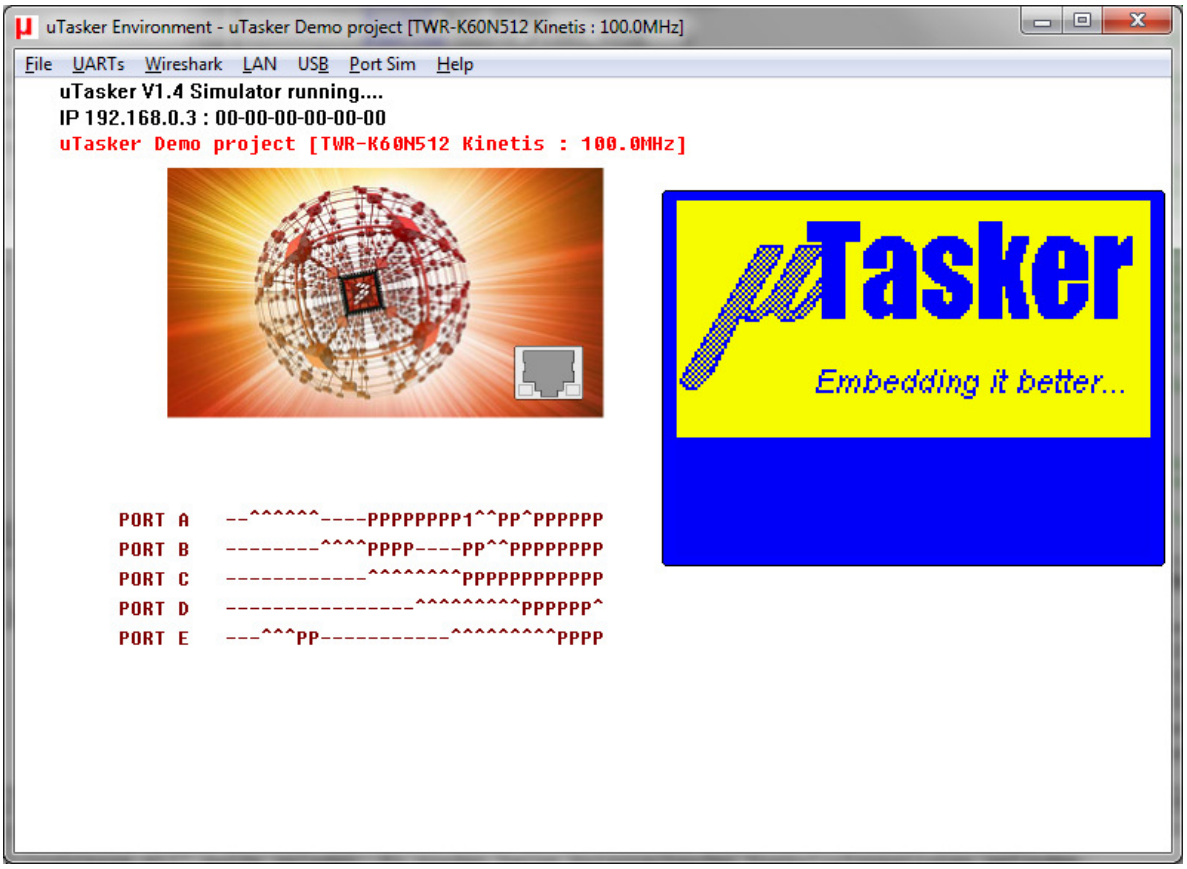


Figure 1 - TFT mono-chrome emulation mode being simulated in the µTasker simulator

In order to work with the TFT display on the TWR-LCD extension board via 16-bit FlexBus it is ***necessary to isolate the line FB_AD9 from the Infra-Red input circuitry***. This can be done by removing R14 and C2 from the board or by cutting the track CMP0_IN0 (between C2 and the back plane). With the circuitry connected the 100nF capacitor C2 otherwise heavily loads the FB_AD9 line and so doesn't allow the bus to operate correctly at normal bus frequencies. With the modification the TFT can be controlled with a maximum FlexBus clock speed of 50MHz without any additional wait-states.

9. USB CDC demo

The USB CDC device demo is described in detail in the document:
http://www.utasker.com/docs/uTasker/uTaskerV1.3_USB_Demo.PDF

When asked for the USB driver, use the file `uTaskerFreescaleVirtualCOM.inf`. This is contained in the SW demo pack or in the project directory `\Applications\uTaskerV1.4\USB`.

The demo document also explains loading new firmware via USB. For details about loading firmware via Web Browser and also configuring the project in CodeWarrior see the following sections of this document.

The guide explains also the µTasker simulator's capability to simulate the USB part of the project, where the following shows the simulator running with enumerated USB.

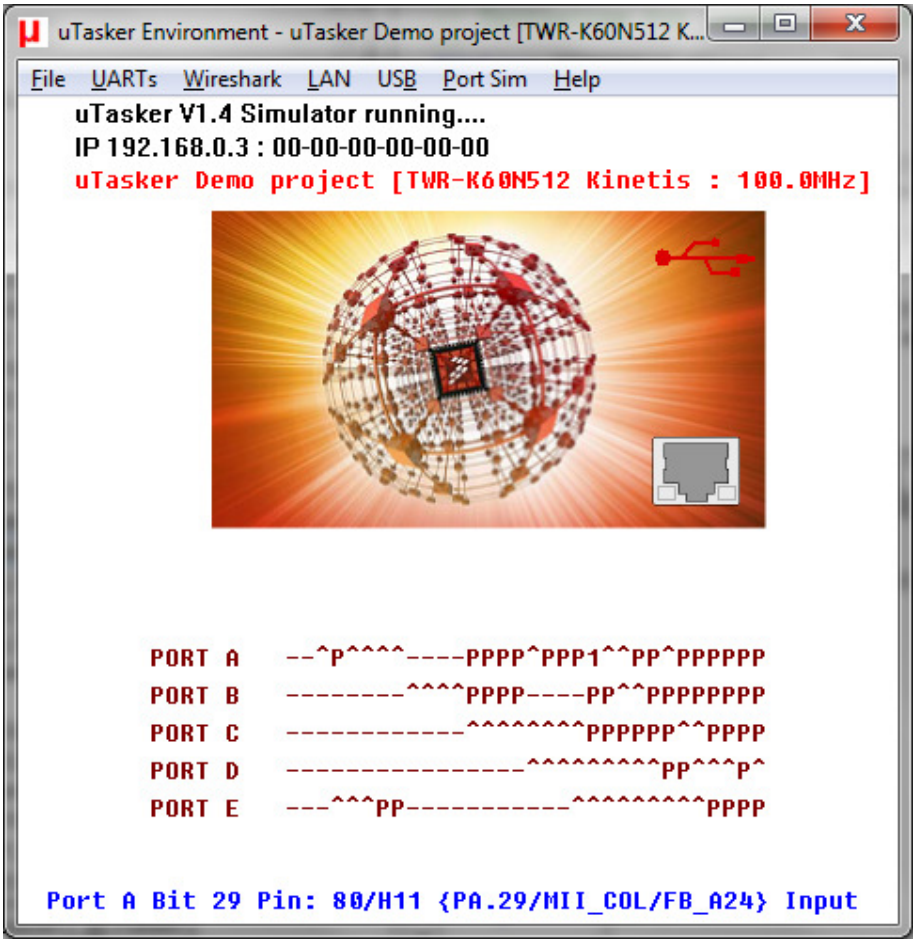


Figure 2 – uTasker simulator running Kinetis, showing enumerated USB state

10. Ethernet

The Ethernet part of the µTasker project is detailed in the Kinetis tutorial:

http://www.utasker.com/docs/KINETIS/uTaskerV1.4_Kinetis_demo.PDF

The tutorial contains a description of working with the µTasker simulator to fully test this functionality as well as compiling with CodeWarrior and GCC. The web pages are discussed in detail but not the method of configuring the project to work with firmware uploads. The following section contains details relevant to this task.

The web pages for the Ethernet demo are contained in the SW demo pack in the sub-directory WebPages. The default IP address if **192.168.0.3** with sub-net mask **255.255.255.0.** If this doesn't suit your test network it can be change in the command line interface via UART or USB:

- Enter the "`Configure LAN Menu`" by entering `1` followed by the ENTER key.
- With the command "`show_config`" the present configuration can be displayed.
- Use "`set_ip_address 192.168.1.125`" to change the IP address to 192.168.1.125 (example IP address – set to a value suitable for your network).
- With "`save`" this is committed to parameter FLASH, which will cause the board to reset and restart.
- Now test that you can ping the new address. If successful, enter into the "`Configure LAN Menu`" again and validate the new setting by commanding "`validate`". This will now commit this new address to be used permanently. Note that the new settings should be validated within a period of three minutes otherwise the temporarily saved ones will be deleted and original ones returned. See the µTasker tutorial for a full explanation of the reasons for this procedure.
- Note also that the board can be configured to obtain its IP configuration from a DHCP server "`set_dhcp enable`". As long as the sub-net settings match the network it is also possible to communicate with the board using its NetBIOS name "`KIRIN3`" (eg. "`ping KIRIN3`") without needing to know its IP address. The NetBIOS name can later be changed in the web server.
- To load the web pages to the target use the bat file `Copy_all.bat`, which is included in the SW demo pack or else can be found in the project directory `\Applications\uTaskerV1.4\WebPages\WebPagesKinetis`. If the IP address of the target has been changed, the IP address setting in the bat file will also need to be modified accordingly. The bat file will automatically transfer all web pages to the target via FTP, after which the board can be browsed using a normal web browser.

The Ethernet section of the demo contains the following main features:

- **TELNET** (command menu is accessible via TELNET and debug output is switched to it when connected)
- **Web Server** with demonstration of dynamic content generation, control of ports, email configuration and transmission via web command (SMPT). Display of user loaded images and firmware upload via HTTP POST.
- **FTP** server for loading web pages and firmware upload.
- **NetBIOS** - can be contacted by name **KIRIN3** (name configurable via web server)
- **DNS** (for SNMP provider resolution)
- Default IP address **192.168.0.3** (subnet mast **255.255.255.0**) – can be configured to other IP configuration via USB or UART menu, including **DHCP**.

11. SD Card

The demo project optionally supports a FAT file system on an SD card. This allows web complicated pages to be served from there, rather than form internal FLASH, and also allows a slide-show to be displayed on the TFT display.

There are two SD card slots available:

- The first is on the Kinetis K60 board and is connected to the SD controller interface of the K60, as well as to SPI1. When the demo is working in SPI mode the operation of this slot is only possible when pins 2 and 7 of the SD card connector are swapped since the SPI1 MOSI/MOSI are otherwise incorrect. **WARNING**: *Newer Tower kits may have the write protect switch of the SD card connected to the reset output (allowing the processor to reset external peripherals in the kit via the port output PTB1). In this case, when the write protect is not set in the SD card it will cause external boards to be held in reset. See the following for more details and work-around:*
http://www.utasker.com/forum/index.php?topic=1737.msg6228#msg6228

- The second is the uSD card slot on the TFT extension board. This is also connected to SPI1 on the Kinetis. To use as interface in SPI mode it is possible that the signals also need to be crossed; in this case check whether the SPI1_SOUT signal is connected to the backplane at B10 or B11. To do this, measure it on the board since it has been found that the board doesn't necessarily correspond with the circuit diagram (Rev. C). SPI1_OUT needs to be connected with B10 and SPI1_SIN with B11.

In case the TWR_LCD board is connected the switch SW1-4 much be set accordingly as shown in the following diagram:

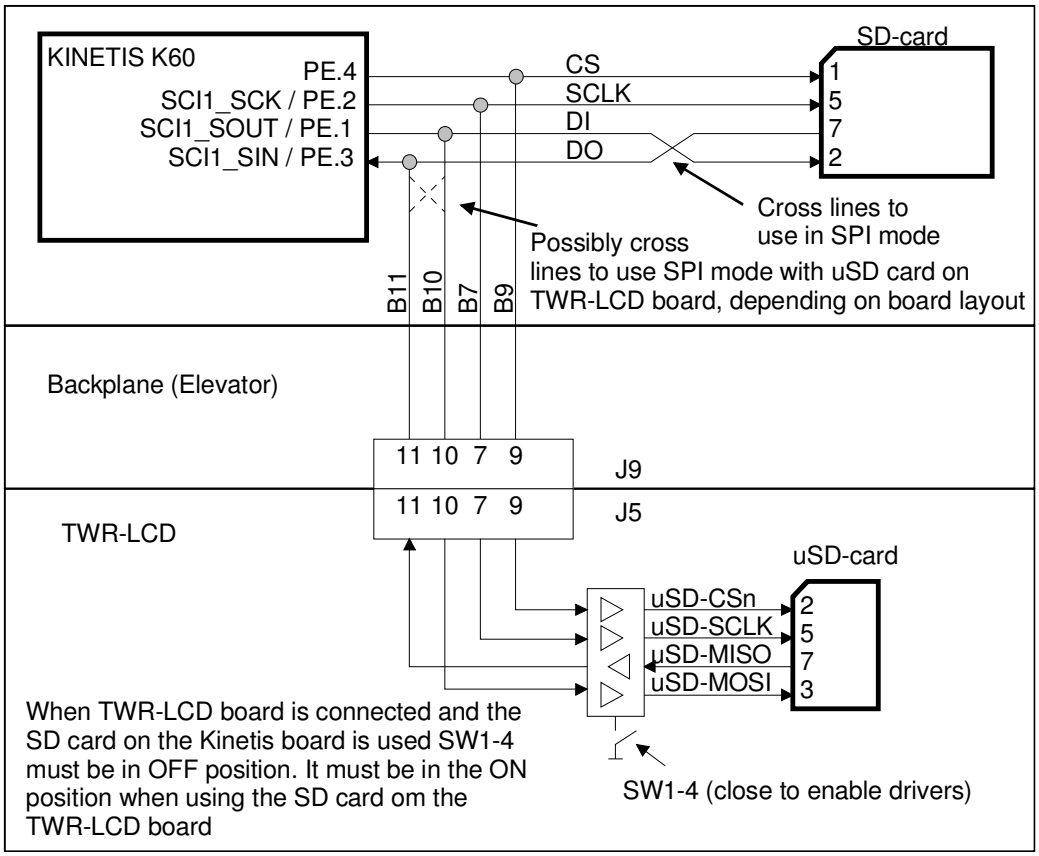


Figure 3 – Connection to SD cards and setting for SW1-4 on the TWR-LCD board

Configuring the µTasker project to support firmware upload

The pre-compiled target file uTaskerV1.4_KINETIS_USB_ETHERNET.S19 from the SW demo pack already contains the µTasker boot loader and the complete application. In this case firmware uploads can already be tested by using the following two files in the SW demo pack:

- z_Upload_3.bin. This is the same application but with a version number **V1.4.003.**. By checking the version number displayed in the web browser or via the command menus (UART, TELNET or USB) it is possible to verify that a firmware upload has been successful.
- z_Upload_2.bin. This is the application contained in the original target file which can be reloaded after testing z_Upload_3.bin or. This then displays the original version number **V1.4.002**.

In order to compile software to upload to the target a different target is required. The corresponding target is KINETIS_BM_ROM (Bare-minimum boot loader target). For full details about working with this and also the boot loader project see the boot loader documents

http://www.utasker.com/docs/uTasker/uTaskerBoot_003.PDF

and

http://www.utasker.com/docs/uTasker/BM-Booloader_for_M5223X.PDF

The target to select in the CodeWarrior boot loader project is M5225X_ROM.

To be noted are following settings for the boot loader project:

- in config.h the defines #define KIRIN3 and #define _M5225X should be active. The first one sets the application start address to 0x1000 (after the first 4k FLASH block in the Kirin3 device) and the second configures the memory map accordingly. *Note that the define KIRIN3 also automatically disables the assembler file workaround for the internal FLASH speculation address qualification incomplete errata of earlier M522XX devices.*
- The demo project uses the following memory map:
  Boot loader occupies 2k (0..0x7ff).
  Application space is 128k (0x1000..0x16aff).
  Parameter system occupies 2 x 2k sectors (0x21000..22fff)
  The file system occupies 372k (0x23000..7ffff), whereby the last 128k (0x60000..0x7fffff) is intended for accepting secure firmware upload

12. Simulating the Kinetis K60 with the SW demo pack executable

In order to allow new users to quickly see how the µTasker simulator operates, an executable is included in the corresponding SW demo pack. This can be found in the sub-directory `\Simulator` and started by simply double-clicking on `uTaskerV1-4-KINETIS-Sim.exe`.

The simulator doesn't need any installation but does require WinPCap to be installed on the PC. If you already have WireShark installed (which also uses it) no further effort is required, otherwise this can be found at http://www.wireshark.org/

The simulator will be displayed as show in the screen shot earlier in this document. It has a default IP address of 192.168.0.1 and opens COM1 on the PC as UART0. This means that by connecting a cross-over cable to COM1 (or by using a virtual com port loop back to it) you can communicate with a terminal emulator (19'600 Baud) and change any settings if needed, as already described in this document and the other guides it links to. When the simulator program is terminated it will always save its complete simulated FLASH contents so that the new settings are valid the next time it is started.

You will need first to configure the PC's NIC (Network Interface Card) that it should use for mapping the Kinetis Ethernet controller to – do this in the "LAN" menu with "Select Working NIC". This setting will also be saved and thus needs only be entered once.

Then use the bat file, as described in the chapter "Ethernet", to transfer the web pages to the simulated device via FTP. After this has been done, connect to its IP address using a standard browser and see the project in operation!

The simulator allows comfortable project development and testing in the VisualStudio environment and the complete project is included as part of the µTasker package.

13. Conclusion

This document has given a quick introduction to compiling and loading the demo software to, and running it on, the **TWR-K60N512** (in combination with the Freescale Tower Kit or stand-alone). As well as giving simple instructions as to how to verify that the software is correctly running is has given links to further documents containing detailed descriptions concerning the use of the USB and the Ethernet parts of the demo, as well as how the project can be configured to enable support of the firmware upload features via USB and Ethernet.

A short guide to using the simulator executable, as included in the SW demo pack, has also been given.

Modifications:
- V0.0 23.12.2011 First version – in progress
- V0.01 9.1.2011 Intermediate version – in progress
- V0.02 14.1.2011 Intermediate version – in progress
- V0.03 25.1.2011 Intermediate version – first developer's version
- V0.04 29.1.2011 Intermediate version – developer's version with new port interrupt details
- V0.05 30.1.2011 Intermediate version – developer's version with new PIT details
- V0.06 4.2.2011 Intermediate version – developer's version with new watchdog and reset details as well as UART restrictions
- V0.07 10.3.2011 Intermediate version – developer's version with new RTC, ADC, DAC, I^2C details
- V0.08 23.3.2011 Intermediate version – developer's version with new SPI details
- V0.09 2.4.2011 Intermediate version – developer's version with new UART RTS/CTS details. DMA uMemcpy(), uMemset() and mass erase command support
-V0.10 10.4.2011 Add notes about using the SD card in SPI mode and the operation of the SPI interface
-V0.11 25.04.2011 Add details about SPI control and timing of the SPI FLASH driver
- V0.12 13.08.2011 Add K40 notes and SLCD
- V0.13 20.10.2011 Add RTC API details and EMAC IP accelerator details, as well as RMII PHY synchronisation
V0.14 03.02.2012: Dual FlexCAN added. FlexTimer PWM notes added.
V0.15 17.03.2012: Added K60F120M, K70F120M and K53N512 boards and details about clock module differences in the high speed parts with FPU. Extended ADC details.
V0.16 27.04.2012 Added notes about different Flash sector sizes in FPU device and DMA details
V0.17 23.6.2012 Added UART3 option on port F for K70. Added notes about RMII_REF_CLK on FPU parts and warning about the SD card write protect pin being connected to the peripheral reset in newer tower card versions.
V0.18 27.10.2012 Added additional DAC details.

Appendix A – Developer's Notes

These developer's notes were created during porting of the µTasker project and peripheral drivers to the Kinetis. They are not intended as replacement for the Kinetis K60 user's manual but instead discuss details that may not necessarily be immediately obvious and also compare the features, operation and techniques required to those of the Coldfire project (a number of peripherals have been inherited from the Coldfire, sometimes with new enhancements).

The K60 devices contain the following amount of RAM and program FLASH:

MK60N256VLQ100	64k	256k
MK60X256VLQ100	64k	256k
MK60N512VLQ100	128k	512k
MK60N256VMD100	64k	256k
MK60X256VMD100	64k	256k
MK60N512VMD100	128k	512k
MK60N256VLL100	64k	256k
MK60X256VLL100	64k	256k
MK60N512VLL100	128k	512k
MK60N256VML100	64k	256k
MK60X256VML100	64k	256k
MK60N512VML100	128k	512k

The K60X versions contain in addition 256k FlexNVM and 4k FlexRAM.

Program FLASH is located from the address 0x00000000 where the reset vector is located – the first long word contains the initial stack pointer value and the second long word the initial program counter value. This is in keeping with the Cortex M4 reset mechanism, which is also the same as used by the Freescale Cortex devices.

The SRAM is centred around 0x20000000. A device with 128k SRAM will therefore have its SRAM from 0x1fff0000 to 0x20000ffff – a device with 64k SRAM 0x1ffff8000..0x20007fff.

Security options are loaded from FLASH on reset. These are held in the area 0x00000400..0x0000040f which means that the correct values need to be stored in this area so that the device is not disabled for debugging, etc. When no security features are to be activated these addresses are programmed to 0x00.

By default the first 1024 bytes of FLASH are used by the reset vector and interrupt vectors. In the case of the µTasker project only the reset vectors are located here and the interrupt vectors are relocated to the start of SRAM – in the case of the K60, 432 bytes of RAM are actually used, after which variables can be stored.

Between the reset vector and the security settings in FLASH there is 1016 bytes of space. Standard functions are forced into this space so that it is generally used efficiently if the compiler doesn't automatically fill it with code.

Clocks – devices up to 100MHz

The module is very flexible and allows various sources to be used to generate a wide range of frequencies for use by the CPU and peripherals.

When the device resets it is clocked by default an internal slow clock which has a frequency of about 32kHz (31.25kHz .. 39.0625 kHz). This is used to derive a frequency 640 times higher by using a FLL (frequency locked loop), meaning that the CPU clock frequency is in the range 20-25MHz.

[*The Flash configuration parameters allow also a 1kHz slow clock default to save power when starting in very low power systems*].

Due to the fact that the µTasker project is used mainly for communication based projects where the clock frequencies need to be within certain tolerances (USB, UART, Ethernet) the initial clocking mode will generally not be suitable since its frequency accuracy is low. Therefore the concentration is on switching from the initial mode to a mode allowing Ethernet to operate with the CPU clocked at its maximum speed for this device of 100MHz. This is accomplished by moving through various intermediate clocking state to arrive at the final state with all circuitry locked ready for final operation.

Note that the clock source much be 50MHz for the Ethernet controller to be able to operate in RMII mode since the clock input is used as clock for this Ethernet module.

The following mode transitions (as defined by the multi-purpose clock generator module) are thus required:

FEI -> FBE -> PBE -> PEE

FEI (FLL Engaged Internal) is the initial state as described above

FBE (FLL Bypassed External) is an intermediate state where the FLL output (initially between 20MHz and 25MHz) is disconnected and the CPU is driven from the external reference clock. This enables the FLL to be configured to generate different frequencies without causing CPU clocking errors to occur. The CPU will in this case be clocked at 50MHz while the next setting is prepared.

PBE (PLL Bypassed External) is an intermediate state where the CPU is clocked from the external reference clock (50MHz in the case of the demonstration). The PLL is now enabled but its output not used, allowing it to be configured ready for the final step.

PEE (PLL Engaged External) is the final state whereby its stabilised output (set to 100MHz) is used to clock the CPU. It is locked to twice the frequency of the external clock (50MHz). The FLL is disabled in a low-power state.

The clocking configuration is set in `app_hw_kinetis.h` by specifying the external clock frequency and the PLL divide and multiplication values.

```
#define EXTERNAL_CLOCK     50000000
// this must be 50MHz in order to use Ethernet in RMII mode

#define CLOCK_DIV          16
// input must be divided to 2MHz..4MHz range (/1 to /25 possible)

#define CLOCK_MUL          32
// the PLL multiplication factor to achieve operating frequency of
100MHz (x24 to x55 possible)
```

From the clock generator output the following main clock signals can be controlled by dividing by 1 to 16 times. This is also necessary in some cases to respect the maximum clocking speed of the individual buses/modules:

- System/Core clock – this can be programmed with a divide value of 1 to obtain the fastest value from a 100MHz PLL output of 100MHz.
- Bus/Peripheral clock – the maximum speed of the peripheral bus is 50MHz and so this requires a divide value of 2 when the PLL clock is 100MHz.
- FlexBus Clock – The maximum value of the FlexBus clock is 50MHz as in the case of the bus clock.
- Flash clock – The maximum Flash clock is 25MHz and so a divide by 4 is required in case of a 100MHz PLL clock output.

The FlexBus and Flash clock speeds should never be higher than the bus clock speed, even when the bus slock speed is lower than the individual maximum frequencies. The same applies to the bus clock frequency, which should never be higher than the system/code clock frequency.

The divide values are set as follows:

```
#define SYSTEM_CLOCK_DIVIDE      1                // 1 to 16 - usually 1

#define BUS_CLOCK_DIVIDE         2                // 1 to 16

#define FLEX_CLOCK_DIVIDE        2                // 1 to 16

#define FLASH_CLOCK_DIVIDE       4                // 1 to 16
```

If any settings are invalid (for example an invalid divide value is attempted or a clock range is out of specification) a compiler check error is generated so that the offending value can be corrected.

The CPU clock frequency generated by the PLL is displayed when running the simulator.

Clocks – devices above 100MHz

Devices with floating point unit (FPU) operate at 120MHz and above. The design of the clock module is however different.
The PLL input has a range of 8MHz to 16MHz. The pre-divider can divide the input by 1 to 8 to obtain a suitable frequency value.
The PLL multiplier has the range x16 to x47
The PLL has an additional output which is twice the PLL output speed, meaning that this output should be set to 240MHz to achieve a system clock of 120MHz. This x2 clock is optionally available for deriving peripheral frequencies from.

Tower TWR-K40X256 clock

This board uses an 8MHz crystal oscillator, rather than an external clock as the K60 tower board uses. This means that its clock initialisation is a bit different because it can't transition from the FEI to FBE state based on an external clock source but must enable the crystal oscillator instead.
The configuration for 96MHz is shown below:

```
#define CRYSTAL_FREQUENCY    8000000     // 8 MHz crystal

#define _EXTERNAL_CLOCK      CRYSTAL_FREQUENCY

#define CLOCK_DIV            4
// input must be divided to 2MHz..4MHz range (/1 to /25 possible)

#define CLOCK_MUL            48
// the PLL multiplication factor to achieve operating frequency of
96MHz (x24 to x55 possible)
```

GPIOs

General purpose I/Os are grouped as 32 bit ports with read, write, clear, set and toggle capability. All ports are disabled (not clocked) out of reset and so they need to first be enabled before they can be used. This involves setting the bit for the particular port in the System Clock Gating Control Register 5 (SIM_SCGC5 in the System Integration Module).

GPIOs are multiplexed with other peripheral signals and can be configured to have specific characteristics. The usual default is to be not connected (disconnected also from the GPIO functionality) or to have a default peripheral function. To be used as digital input or output they must be connected by setting each individual port bit control accordingly to GPIO function. Each GPIO input can be configured to have no pulls, pull-up or pull-down, with either a 2MHz passive input filter or a optional digital glitch filter. Each GPIO output can be configured to be push-pull or open-drain, with full-speed or limited slew rate, with full drive or limited drive strength.

By setting a lock bit the port pin configuration can be protected against further modification until the next system reset.

To cater for the need to control each individual pin the µTasker port macro has been extended to automatically clock the port and include passing the characteristics of the pins when they are configured. The following shows a single GPIO being configured as an output with the initial value ‘1’ as open-drain output with limited slew rate and high drive strength.

```
_CONFIG_DRIVE_PORT_OUTPUT_VALUE(A, PORTA_BIT11, PORTA_BIT11,
(PORT_ODE | PORT_SRE_SLOW | PORT_DSE_HIGH));
```

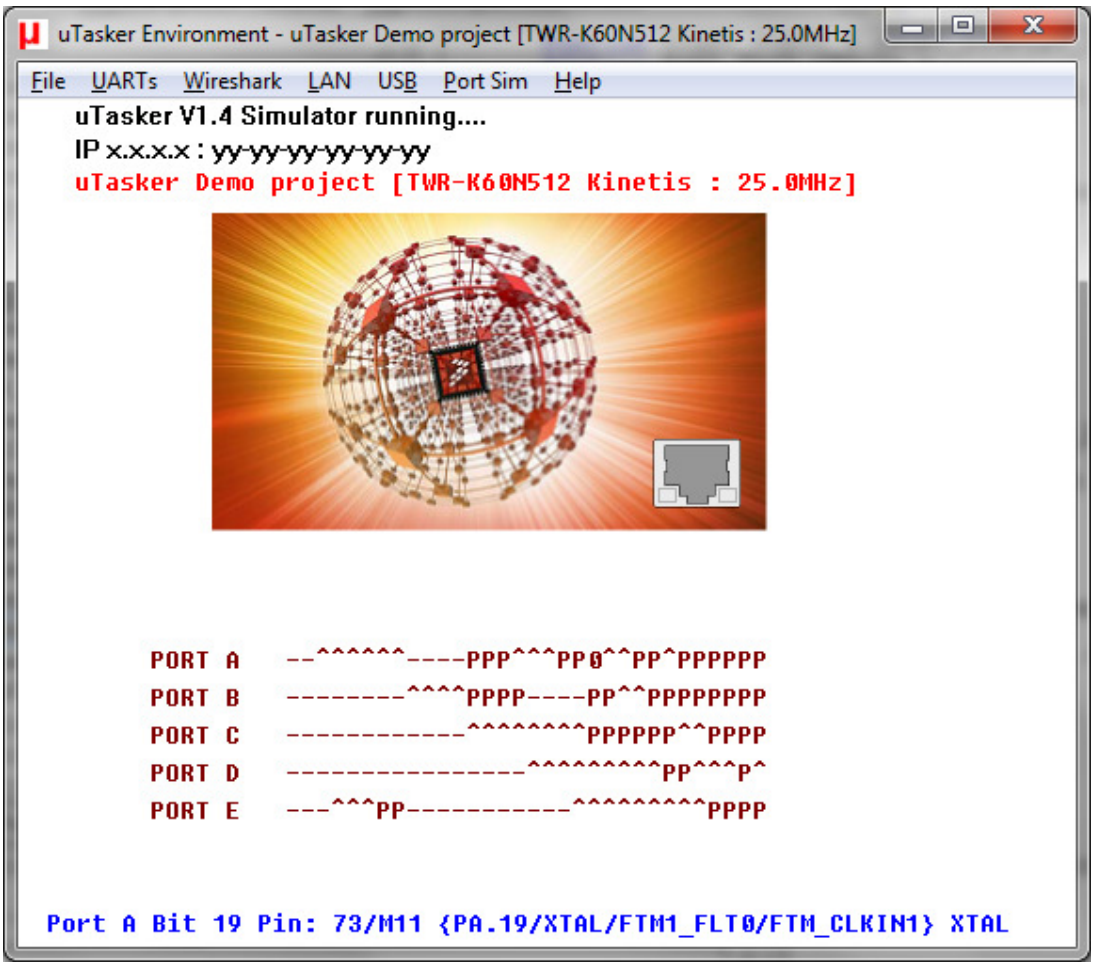
When multiple pins are defined each of them are set to have the same configuration characteristics. The macros that include the full configuration are:

```
_CONFIG_PORT_OUTPUT(ref, pins, chars)

_CONFIG_PORT_INPUT(ref, pins, chars)

_CONFIG_DRIVE_PORT_OUTPUT_VALUE(ref, pins, value, chars)
```

The µTasker simulator simulates the port states out of reset so that the default function of each pin can be simply identified. Unconnected pins are displayed as inputs but are not actually connected to the GPIO input function. Pins defaulting to a peripheral function are displayed as ‘P’ and their exact function can be seen by hovering the mouse over the port bit.



Simulator showing the state of pins out of reset (PA.11 has been subseqeunty set as output). Hovering the mouse over PA.19 shows its possible functions and that its default is XTAL

To configure a peripheral function the macro `_CONFIG_PERIPHERAL(port, pin, function)` can be used. For example:

```
_CONFIG_PERIPHERAL(A, 15, PORT_MUX_ALT4);
```

This sets the alternative function 4 to port A, bit 15. In this case it is `MII0_TXEN` on PA.15. Alternatively the following can also be used when a corresponding peripheral multiplexing define is available.

```
 _CONFIG_PERIPHERAL(A, 15, PA_15_MII0_TXEN);
```

Furthermore the characteristics of digital peripherals can also be controlled at the same time as their multiplexing is configured:

```
_CONFIG_PERIPHERAL_VALUE(B, 17, (PORT_MUX_ALT5 | PORT_DSE_HIGH |
PORT_PS_DOWN_ENABLE));
```

This shows the alternative 5 peripheral (`FB_AD16`) being set to port B-17 with high strength and pull-down enabled

It is important to note that the ports need to be powered up to enable setting peripheral functions on them. For this reason the peripheral select macros automatically power on the ports of the peripheral pins selected.

See the following forum post for a fast port configuration macro added to the project
http://www.utasker.com/forum/index.php?topic=1664.0

UARTS

The K60 has 6 UARTs with RXD, TXD, RTS and CTS signals.

The UARTs support infrared and ISO 7816 T0/T1 protocols.

UARTs 0 and 1 are clocked from the core/system clock, which allows highest performance potential. The other UARTs are clocked from the bus clock.

The UARTs can usually be mapped to various pins on the device. The following table shows the possibilities.

UART0	**UART0_TXD**	**UART0_RXD**	**UART0_RTS**	**UART0_CTS**
UART0_A_LOW	Port A2 (Alt 2)	Port A1 (Alt 2)	Port A3 (Alt 2)	Port A0 (Alt 2)
default	Port A14 (Alt 3)	Port A15 (Alt 3)	Port A17 (Alt 3)	Port A16 (Alt 3)
UART0_ON_B	Port B17 (Alt 3)	Port B16 (Alt 3)	Port B2 (Alt 3)	Port B3 (Alt 3)
UART0_ON_D	*Port D7 (Alt 3)	Port D6 (Alt 3)	Port D4 (Alt 3)	Port D5 (Alt 3)
UART1	**UART1_TXD**	**UART1_RXD**	**UART1_RTS**	**UART1_CTS**
default	Port E0 (Alt 3)	Port E1 (Alt 3)	Port E3 (Alt 3)	Port E2 (Alt 3)
UART1_ON_C	Port C4 (Alt 3)	Port C3 (Alt 3)	Port C1 (Alt 3)	Port C2 (Alt 3)
UART2	**UART2_TXD**	**UART2_RXD**	**UART2_RTS**	**UART2_CTS**
Default	Port D3 (Alt 3)	Port D2 (Alt 3)	Port D0 (Alt 3)	Port D1 (Alt 3)
K70 – UART2_ON_E	Port E16 (Alt 3)	Port E17 (Alt 3)	Port E19 (Alt 3)	Port E18 (Alt 3)
K70 – UART2_ON_F	Port F14 (Alt 4)	Port F13 (Alt 4)	Port F11 (Alt 4)	Port F12 (Alt 4)
UART3	**UART3_TXD**	**UART3_RXD**	**UART3_RTS**	**UART3_CTS**
default	Port E4 (Alt 3)	Port E5 (Alt 3)	Port E7 (Alt 3)	Port E6 (Alt 3)
UART3_ON_B	Port B11 (Alt 3)	Port B10 (Alt 3)	Port B8 (Alt 3)	Port B9 (Alt 3)
UART3_ON_C	Port C17 (Alt 3)	Port C16 (Alt 3)	Port C18 (Alt 3)	Port C19 (Alt 3)
K70 – UART3_ON_F	Port F8 (Alt 4)	Port F7 (Alt 4)	Port F9 (Alt 4)	Port F10 (Alt 4)
UART4	**UART4_TXD**	**UART4_RXD**	**UART4_RTS**	**UART4_CTS**
default	Port E24 (Alt 3)	Port E25 (Alt 3)	Port E27 (Alt 3)	Port E26 (Alt 3)

UART4_ON_C	Port C15 (Alt 3)	Port C14 (Alt 3)	Port C12 (Alt 3)	Port C13 (Alt 3)
UART5	**UART5_TXD**	**UART5_RXD**	**UART5_RTS**	**UART5_CTS**
default	Port E8 (Alt 3)	Port E9 (Alt 3)	Port E11 (Alt 3)	Port E10 (Alt 3)
UART5_ON_D	Port D9 (Alt 3)	Port D8 (Alt 3)	Port D10 (Alt 3)	Port D11 (Alt 3)

*UART0_TXD on Port D7 is special since its drive strength can be programmed by CMTUARTPAD in SIM_SOPTS.

The µTasker UART driver supports 6 channels and defaults to the pin-outs as shown in the "default" rows. By activating the alternative defines, for UARTs with multiple positions, the set of combinations can be configured accordingly.

The UARTs support 10 and 11 bit characters. That means that they can operate in 8 bit character mode with or without parity but plus one stop bit:

- start bit, 8 bit character, one stop bit)
- start bit, 8 bit character, parity bit, one stop bit)

or 7 bit character mode with parity

- start bit, 7 bit character, parity bit, one stop bit)

In addition an address bit can be used (generally known as multi-drop or 9 bit mode).

When in 7 bit character mode the parity bit is received as 8th bit in the receive data register and so needs to be masked out in software to avoid corruption.

This means that there are some restrictions as to what formats are possible since 1.5 or 2 stop bits cannot be configured and also 7 bit character mode without parity is not supported.

RTS/CTS operation

The UARTs support various modem modes of operation. The two most important are the control of RTS when sending on a bidirectional line (like RS485), whereby the RTS should drive the transmitter transceiver (either positive or negative), and HW flow control, whereby the RTS and CTS are used to control the flow of data between the two ends [transmitter can only send when CTS is asserted].

To control the RTS line to envelope the data transmission characters a mode is available. By enabling the bit TXRTSE in the UARTx_MODEM the UART automatically asserts the RTS line when there is data in the output buffers and negates it automatically after the stop-bit of the last data byte has been sent. In addition the polarity of the RTS line can be controlled using the TXRTSPOL bit in the same register.

The RS485 mode can be controlled by the driver routine:

```
fnDriver(SerialHandle, (CONFIG_RTS_PIN | SET_RS485_MODE |
SET_RS485_NEG), 0);
```

When not in RS485 mode the RTS signal can be controlled by software by using

```
fnDriver(SerialHandle, (SET_RTS), 0);
```

and

```
fnDriver(SerialHandle, (CLEAR_RTS), 0);
```

In this case the RTS line must have been configured as non-RS485. The UART negates the RTS line when CLEAR_RTS is commanded by disabling its control based on the state of the receiver FIFO. When SET_RTS is commanded the UARTs automatic RTS mode is selected which then controls the state of the RTS line based on its FIFO. If the FIFO becomes critically full it will automatically negate the signal but will otherwise assert it. This means that the UART will automatically protect its own FIFO from overruns as long as HW flow control is supported by the remote transmitter.

The µTasker serial driver will also automatically control the RTS line when the input buffer becomes critically full.

To perform HW flow control in the opposite direction, the CTS line can be configured to automatically control the transmitter. This mode is automatically configured when the HW flow control mode is configured [`tInterfaceParameters.Config |= RTS_CTS`].

The bit `TXCTSE` set in `UARTx_MODEM` is set so that the UART automatically suspends transmission when the input is negated. No CTS interrupt is generated so the control is purely performed in hardware. The state of the CTS line cannot be specifically requested.

UART DMA transmission is supported when the define `SERIAL_SUPPORT_DMA` is enabled in the project. In this case each individual UART can be configured for either DMA or interrupt operation when the UART is configured:

```
tInterfaceParameters.ucDMAConfig = UART_TX_DMA;  // use DMA for transmission
                                                    on the UART
SerialPortID = fnOpen(TYPE_TTY, FOR_I_O, &tInterfaceParameters);
                                                 // configure and open UART interface
```

DMA channels triggered by peripherals use the DMA MUX. When peripheral DMA is in operation the DMA MUX is automatically enabled and configured to suit the peripheral. The DMA channel used by each peripheral is configurable in `app_hw_kinetis.h` in the DMA settings block, whereby the channels must be unique for all used ones.

```
#define DMA_UART0_TX_CHANNEL       3   // use this DMA channel when using UART 0
                                       for transmission driven by DMA
#define DMA_UART1_TX_CHANNEL       4   // use this DMA channel when using UART 1
                                       for transmission driven by DMA
#define DMA_UART2_TX_CHANNEL       5   // use this DMA channel when using UART 2
                                       for transmission driven by DMA
#define DMA_UART3_TX_CHANNEL       6   // use this DMA channel when using UART 3
                                       for transmission driven by DMA
#define DMA_UART4_TX_CHANNEL       7   // use this DMA channel when using UART 4
                                       for transmission driven by DMA
#define DMA_UART5_TX_CHANNEL       8   // use this DMA channel when using UART 5
                                       for transmission driven by DMA
```

The priority of the channel used is set to the the same level as the channel's number (eg. if DMA channel 4 is used for a particular UART transmitter its DMA priority is also 4).

If it is necessary to change priorities of the channels this should be performed by using a set of channel priority definitions which will be set at initialisation:

```
#define DMA_CHANNEL_0_PRIORITY  15
#define DMA_CHANNEL_1_PRIORITY  14
#define DMA_CHANNEL_2_PRIORITY  13
..
#define DMA_CHANNEL_14_PRIORITY  1
#define DMA_CHANNEL_15_PRIORITY  0
```

Changing priorities after initialisation is not supported due to the risks involved but swapping channels used (and so priority settings) is possible at run-time. It is to be noted that any DMA operation that occurs when priority levels are being changed and there is a conflict (two channels with same priority level) this will lead to an error and suspend the DMA attempt. Since it is generally not realistic to change priorities safely without disabling all DMA operation temporarily (and possibly losing some peripheral data) this is avoided.

Reception via DMA is less common due to the fact that there is no indication of individual characters being receive d(without polling) but can be enabled using

```
tInterfaceParameters.ucDMAConfig |= UART_RX_DMA;  // use DMA for reception
                                                    on the UART
```

Each complete buffer received results in a message with its character content to be informed of. Half-buffers are used if the flag `UART_RX_DMA_HALF_BUFFER` is set as well when the interface is opened/configured.

Again each UART channel used needs to be defined to a particular DMA channel

```
#define DMA_UART0_RX_CHANNEL       9   // use this DMA channel when using UART 0
                                       for reception driven by DMA
#define DMA_UART1_RX_CHANNEL       10  // use this DMA channel when using UART 1
                                       for reception driven by DMA
#define DMA_UART2_RX_CHANNEL       11  // use this DMA channel when using UART 2
                                       for reception driven by DMA
#define DMA_UART3_RX_CHANNEL       12  // use this DMA channel when using UART 3
                                       for reception driven by DMA
#define DMA_UART4_RX_CHANNEL       13  // use this DMA channel when using UART 4
                                       for reception driven by DMA
#define DMA_UART5_RX_CHANNEL       14  // use this DMA channel when using UART 5
                                       for reception driven by DMA
```

Ethernet Controller

Warning: due to a conflict in the sharing of the MII0_RXER and JTAG_TRST lines MII0_RXER (optional line) cannot be used together with a JTAG debugger. This is not a problem when debugging using the SWD interface, such as the inbuilt debugger on the tower kit. When using JTAG the MII0_RXER function should not be used and can be disabled by setting the define JTAG_DEBUG_IN_USE_ERRATA_2541.

The Ethernet controller is basically compatible with the one in the Coldfire. It has the same control register set and MIB counters.

In addition it has IEEE1588 support and a capture/compare block.

The tower kit supports RMII mode of operation at the TWR-SER board and the K60 board is defined for operation in this configuration. This means that the following signals are required for Ethernet operation (see also the discussion about the clock setting in the main text in the section about recommended jumper configurations):

- MII_MDIO (PB0 – ALT 4) maintenance connection to PHY – serial data line
- MII_MDC (PB1 – ALT 4) maintenance connection to PHY – serial clock
- RMII_CRS_DV (PA14 ALT 4)
- RMII_RXD[0] (PA13 ALT 4)
- RMII_RXD[1] (PA12 ALT 4)
- RMII_RXER (PA5 ALT 4)
- RMII_TXD[0] (PA16 ALT 4)
- RMII_TXD[1] (PA17 ALT 4)
- RMII_TXEN (PA15 ALT 4)
- RMII_REF_CLK – this is taken from the internal clock OSCERCLK, which is a buffered signal from the EXTAL input (oscillator or crystal). This signal must be 50MHz for the RMII Ethernet mode to operate.
  The signal is also available as dedicated input on some device, such as the FPU parts, which means that the main oscillator is not required to be 50MHz.

The define `ETHERNET_RMII` should be used.

The following differences exist in comparison to the M52259 Kirin3 (as reference)

1) It is not possible to write to the MIB block. When writes are attempted a hard fault occurs (to clear the MIB a memcpy() call is made with the Kirin3). The MIBC register contains a new bit called MIB_CLEAR which can be used to command a clear instead.
2) K60 can operate in RMII mode at 100MHz.
3) The CRC32 can be automatically stripped when receiving Ethernet frames.
4) The Ethernet buffer descriptors operate in big-endian mode. This means that their content flag format is byte-swapped in comparison to when operating on a Coldfire. In addition the content length (saved as short word in the descriptors) needs to be converted as do any memory pointers in the descriptors.

5) The Ethernet buffer descriptors operate in legacy mode by default, meaning that Coldfire code (with the exception of little-endian conversions) operates without any large modifications. The buffer descriptors can also work in extended mode – this mode has not yet been implemented.
6) The Kinetis EMAC automatically pads transmitted frames to 64 byte length

An interesting feature in the EMAC is the new support for IP checksum offloading. This allows the HW to automatically check or generate checksums for IP, ICMP, TCP and UDP frames (IPv4 and IPv6), which make the operation much more efficient than when SW needs to do this – see the following for some figures showing the loading of check sum calculation: http://www.utasker.com/docs/uTasker/uTaskerBenchmarks.PDF

To enable check sum offloading in transmission IP headers enable
`IP_TX_CHECKSUM_OFFLOAD`

To enable check sum offloading in transmission IP protocol payloads enable
`IP_TX_PAYLOAD_CHECKSUM_OFFLOAD`

Enable both to use both transmission offloading features.

To automatically discard frames received with IP header or protocol checksums the defined `IP_RX_CHECKSUM_OFFLOAD` is enabled. Since no frames will be received with such errors the IP software also doesn't need to verify them.

PHY and RMII

It is to be noted that RMII mode of operation requires more effort to match the Ethernet controller's mode to the present line characteristics. Since the PHY is the part actually monitoring the line and informing of line changes (link up, link down, link characteristic changes) it is important that the PHY and Ethernet work together. Essentially all link state changes are reported via its interrupt line and the Ethernet driver can change the Ethernet controllers mode settings accordingly. This is performed in the Ethernet driver to ensure that the RMII mode of the Ethernet controller (10M or 100M, half-duplex or full-duplex) is each time synchronised to that which is being detected by the PHY. The Freescale tower kit operates in RMII mode although the Kinetis itself can be connected in MII mode, whereby the adaptation is then controlled automatically by the interface itself.

Communication between the PHY and the Kinetis takes place via the MII management channel. The serial speed is usually 2.5MHz but it is run at 800kHz in the tower kit since the lines are long (they traverse the backplane) and the weak pull-up on the data line, together with the backplane capacitance otherwise result in a speed limitiation.

When there is no communication between the PHY and the Kinetis the MII management channel's clock is stopped wherever possible to reduce interference and save (a small amount of) power.

FLASH

The X parts contain FlexNVM and FlexRAM. The N parts contain just program FLASH which can however also be programmed by the application. Since the most common parts are the N parts these will be concentrated on here.

The FLASH controller allows simple programming and erasing of FLASH. Flash programming can be performed on long words (cumulative programming of single long words is expressly forbidden and so is to be avoided) and larger buffer sizes too.

FLASH can be erased in sectors (2k in size for non-FPU types and 4k for FPU types) or in blocks (256k in 512k parts). The FLASH granularity is thus 2k resp. 4k (smallest erasable block).

Programming of a long word takes typically 50us. Erasing a sector takes typically 20ms (maximum 100ms). A block erase takes typically 160ms (maximum 800ms).

Long word programming is however only possible on non-FPU parts because the FPU parts use “phrase” programming instead (based on 8 bytes)

The program FLASH is divided into two banks (blocks) and it is possible to erase/program in one bank when code is being executed from the other. It is not possible to execute code from a bank in which a write or erase operation is being performed and so generally the programming routine runs from RAM and interrupts need to be avoided if their handlers can cause code in the programming area to be called.

The mass erase command `fnMassEraseFlash()` is supported by using the “ERASE ALL BLOCKS COMMAND”. This deletes all types of internal memory and also removes any block protection security and so can be used to unlock general security settings.

Reset and Watchdog

The mode controller can be read to determine the reason of the last reset. Reasons range from power on reset, external reset, wakeup reset, watchdog reset to loss-of-clock reset.

The routine `extern unsigned char fnAddResetCause(CHAR *ptrBuffer)` can be used to fill a string buffer with the reset cause and is displayed on the administrator web page when the web server demo is active.

The internal watchdog operates out of reset and is clocked by default from the external system clock and it must be either reconfigured or stopped within a short period of time of 256 bus clock cycles after reset. It is therefore necessary to perform this activity before the initialisation code performs variable initialisation to avoid a watchdog reset taking place.

The watchdog is set to a value of 2s by default in the µTasker demo project and can be disabled by holding SW1 on the TWR-K60N512 board. It is triggered every 200ms by the watchdog task when the system is operating normally. When triggering the watchdog interrupts are temporarily disabled so that the trigger sequence, that much be completed within 20 bus cycles, is not disturbed. The trigger sequence involves writing the sequence 0xa602 and 0xb480 to the refresh register.

The Kinetis includes a further watchdog module called the external watchdog monitor which is designed to reset external circuitry in case of a software error. It doesn't reset internal circuitry since this is the job of the internal watchdog.

The user can command a reset of the board by using the routine `extern void fnResetBoard(void)`. This uses the Cortex's reset request to initiate the reset the fact is signalled in the mode controller's reset cause register after the next reset.

Important notes about debugging: Due to the fact that the watchdog configuration needs to be performed quickly after a reset it is recommended to allow the debugger to run to main() when it connects. This allows it to immediately start the processor and allow the code to quickly configure the watchdog as required – even with active watchdog configured it can be set up to stop when the debugger pauses execution (this is the default configuration in the project).

If the start-up code needs to be stepped through, the watchdog unlocking sequence should be avoided since it will result in a reset due to the fact that the stepping has taken too long to reach the unlocking location. See also the following forum post for details of speeding up the code to reach the watchdog unlocking sequence:
http://www.utasker.com/forum/index.php?topic=1664.0

Interrupts

The size of the interrupt vector table depends on the number of interrupts that are possible – the K60 requires 432 bytes to hold its maximum total 216 interrupt vectors. If interrupt vectors located higher in the vector table are not actually used less space would in fact be needed, but the 432 bytes space is reserved to accommodate all possibilities.

Rather than use the interrupt vectors directly from fixed locations at the start of FLASH they are entered into SRAM only when needed. The first 432 bytes of SRAM are reserved for this use, meaning that linker script files allow the first used address to be `0x1fff01b0`. The Cortex M4 vector table offset register is set to `0x1fff0000` so that interrupt vectors are taken from there.

This strategy allows interrupt handlers to be exchanged in SRAM during operation and also is suited to systems operating with a boot loader since its interrupt vectors cannot be fixed at the start of FLASH, which is occupied by the boot start-up code.

It is to be noted that not all Kinetis parts have compatible interrupt vector tables. For example, low power parts that are specified for 50MHz have less interrupts and a correspondingly smaller table with interrupt vector IDs at different location to its 75MHz or 100MHz parts.

Port Interrupts

Each GPIO can be used as an interrupt line (the Kinetis doesn't have IRQ0, IRQ1, IRQ2… as the Coldfire has).

The port interrupt support is enabled by the define `SUPPORT_PORT_INTERRUPTS` (in `app_hw_kinetis.h`). This allows individual interrupt handler routines to be defined to each of the port pins, whereby the Kinetis has one interrupt vector for each of the ports available (port A to port E). If any ports are not required for interrupt generation they can be specifically excluded from the code by setting `NO_PORT_INTERRUPTS_PORTA`, `NO_PORT_INTERRUPTS_PORTB`, etc. as appropriate.

The following shows how a port interrupt can be simply defined:

```
INTERRUPT_SETUP interrupt_setup;
interrupt_setup.int_type      = PORT_INTERRUPT;     // identifier to configure port interrupt
interrupt_setup.int_handler   = fnInterruptHandler; // handling function
interrupt_setup.int_priority  = PRIORITY_PORT_INT;  // interrupt priority level
interrupt_setup.int_port      = PORTA;              // the port that the interrupt input is on
interrupt_setup.int_port_bits = PORTA_BIT27;        // the IRQ input connected
interrupt_setup.int_port_sense = (IRQ_LOW_LEVEL | PULLUP_ON);
                                                    // interrupt is to be low level sensitive
fnConfigureInterrupt((void *)&interrupt_setup);     // configure interrupt
```

This shows a level sensitive interrupt being configured on port A, bit 27. Also a pull-up resistor is being configured in the port input.

The interrupts are flexible and can be falling edge, rising edge or both-edge sensitive. They can also be level high or level low sensitive. The type can be selected by defining `IRQ_FALLING_EDGE`, `IRQ_RISING_EDGE`, `IRQ_BOTH_EDGES`, `IRQ_LOW_LEVEL` or `IRQ_HIGH_LEVEL` as `interrupt_setup.int_port_sense` value.

An interrupt can be set to multiple port pins by defining more than one input belonging to the port as `interrupt_setup.int_port_bits` value. For example, `interrupt_setup.int_port_bits = (PORTA_BIT27 | PORTA_BIT18 | PORTA_BIT10);` will cause the same handler to be called for matching states or changes on any of these pins.

Since the port interrupt interface handles clearing of the interrupt source the user handler doesn't need to be concerned with these details.

Periodic Interrupt Timers (PIT)

The K70 has 4 PITs. These are however not the same as the ones in the Coldfire.

Improved features include 32 bit counters that can also trigger DMA transfers or ADC conversions. Each of the PITs is realised as a down counter which are clocked from the peripheral bus clock.

The following example shows how a PIT (Pit 1 shown) can be simply configured for a single shot interrupt:

```
PIT_SETUP pit_setup;                          // interrupt configuration parameters
pit_setup.int_type = PIT_INTERRUPT;
pit_setup.int_handler = test_timer_int;       // test a single shot timer
pit_setup.int_priority = PIT1_INTERRUPT_PRIORITY;
pit_setup.count_delay = PIT_US_DELAY(3245);   // 3245us delay
pit_setup.mode = PIT_SINGLE_SHOT;             // one-shot interrupt
pit_setup.ucPIT = 1;                          // use PIT1
fnConfigureInterrupt((void *)&pit_setup);     // enter interrupt for PIT1 test
```

When the timer has fired it will be automatically stopped. If there are no other PITs active the PIT module will also be automatically powered down.

To start a periodic interrupt the mode `PIT_PERIODIC` is used instead. In this case it will run until stopped with the mode command `PIT_STOP`. Again, if there are no other PITs operating when the PIT is stopped the module will be automatically powered down.

Since the PITs have 32 bit counters and are clocked at the bus rate, with maximum speed of 50MHz, the maximum period that they can be set to at the highest bus speed is about 85s.

FlexBus

The FlexBus has 6 independent chip selects and is thus almost identical to the FlexBus in the M520X Coldfires (also the Kirin3 M5225X, which however has only two independent chip select lines).

It has one additional register called the Chip Select Port Multiplexing Control Register which controls the multiplexing of some of the available signals, depending on the bus interface type required.

The FlexBus is intended mainly for multiplexed bus operation whereby the data and address bus widths are part dependent. The FlexBus memory range is between 0x60000000 and 0cdfffffff.

The K60 supports FB_AD0..FB_AD31 (a full 32 bit multiplexed bus) and FB_A16..FB_A29 as non-multiplexed address bus. The bus lines used are selected by the port peripheral functions alone.

The TWR-K60N512 can control the TFT display on the TWR-LCD extension module via a 16 bit multiplexed bus, whereby the lines FB_AD0..FB_AD15 are used as data bus (note that the 16 bit accesses can be controlled to occur on either FB_AD0..FB_AD15 or FB_AD16..FB_AD31; in comparison the Mini-FlexBus in the Kirin3 has only FB_AD..FB_AD15.

The line FB_AD16 is used as an address line to select between command and data access and the control lines FB_CS0 and FB_RW are the only other signals required for complete control.

DMA Controller

The K60 has a 16 channel extended DMA controller.

When the define `DMA_MEMCPY_SET` is enabled the `uMemcpy()` and `uMemset()` routines use a single channel of the eDMA controller to perform the memory transfers. This is several times faster than performing the copy via the CPU. The channel is given the lowest priority and is allowed to be temporarily stalled by higher priority DMA channels while operating. Should an IRQ perform a copy based on these functions any active copy is protected and the IRQ will then use a conventional copy instead.

The DMA channel used for the memory copy routines is defined by

```
#define DMA_MEMCPY_CHANNEL 15
```

The priority of the used channel is set to 0 (lowest) and the priority of the used channel (when not 0) is set to channel 0 so that no two channels have the same priority, which is otherwise illegal.

It is to be noted that only the first 4 DMA channels support periodic transfers and so must be used together with PITs if used in this mode.

All devices have DMA channels 0..15. The devices with FPU have a second DMA group with channels 16..31. Since the groups' DMA channels are prioritised between 0 and 15 (15 is highest priority) and no two channels may have the same priority also the groups have their own priorities that much ne unique. The group 0 (DMA channels 0..15) are set to group priority 0 and the group 1 (DMA channels 16..31) are set to group priority 1 in the µTasker project to ensure that they don't collide. This means that the user can assign DMA channels from group 1 (when using devices with this) if they should have higher priorities than those in group 0.

RTC

The K60 has a low power Real Time Clock module with an independent power supply (VBAT), a 32.768kHz oscillator and optional 1Hz square wave output. The 32.768kHz clock output can also be supplied to the rest of the chip if required. The crystal can be calibrated between 0.12 ppm and 3906 ppm.

The RTC is not a RTC which counts second, minutes, hours, days, etc. but is a 32 bit seconds counter. This means that conversion to actual time/dates needs to be handled in software, as are such details about the number of days in a months and when a leap-year has to be considered.

Furthermore the power to the RTC is always taken from the VBAT input, also when the processor is powered up. This means that there is battery drain even when the processor is powered if this is not solved by external hardware and the RTC cannot be used without VBAT, even if it doesn't need to operate when normal power has been removed (of course this is not a problem since VBAT can also be simply connected to the general power supply in such a usage case).

The VBAT voltage level should be between 1.71V and 3.6V. The typical current consumption with active 32.768 kHz oscillator operating is 550nA, which is a good value for an internal RTC. However there are no additional user-registers that can be retail values in the module, which are often available in RTC modules.

Software should not try to access the RTC operating registers when the VBAT is not connected since this will result in a bus error.

There are 4 pins that can be assigned to the RTC.

EXTAL32, XTAL32 - for 32.768kHz crystal connection

RTC_CLKOUT - for 1Hz square wave output

RTC_WAKEUP – open-drain, active low output to wake up external devices when the RTC interrupt occurs. This asserts when the device is powered down and can thus be used to start a system out of this mode at a pre-defined point in time *This output is not available on all device types*.

The RTC interrupt can be used to wake the chip from any low power mode.

The K40 doesn't seem to implement the seconds interrupt in the RTC. *A seconds interrupt can however be achieved by using an alarm interrupt always set to the next second.*

Note that the RTC crystal should be positioned close to the device and its signals should be short and protected against cross-talk from other signal lines. The start-up time is typically 1s and this time should be waited after initial activation before using the RTC or its clock at other modules.

The µTasker project implements a RTC API including Gregorian date/time from the 1s interrupt. It emulates the RTC in some Coldfire to additionally allow minutes, hours, days, stop-watch interrupts as well as date/time alarm interrupt.More information is available at *http://www.utasker.com/forum/index.php?topic=1656.0*

ADC

The Analogue to Digital Converter (ADC) module used in the Kinetis devices supports up to 4 differential analogue inputs and up to 24 single-ended analogue inputs. Based on linear successive approximation it offers up to 16-bit resolution (note that several resolution modes are possible and the low power ADC achieves up to 14 bit of actual accuracy).

The ADC supports a self-calibration mode.

Using its compare registers it can compare ADC results and also ranges.

The following table shows the pins that are used by the ADCs in the K60 and also multiplexed with ports:

ADC name	Port	Function
ADC0_SE10	PTA7	ALT 0
ADC0_SE11	PTA8	ALT 0
ADC1_SE17	PTA17	ALT 0
ADC0_SE8	PTB0	ALT 0
ADC0_SE9	PTB1	ALT 0
ADC0_SE12	PTB2	ALT 0
ADC0_SE13	PTB3	ALT 0
ADC1_SE10	PTB4	ALT 0
ADC1_SE11	PTB5	ALT 0
ADC1_SE12	PTB6	ALT 0
ADC1_SE13	PTB7	ALT 0
ADC1_SE14	PTB10	ALT 0
ADC1_SE15	PTB11	ALT 0
ADC0_SE14	PTC0	ALT 0
ADC0_SE15	PTC1	ALT 0
ADC0_SE4	PTC2	ALT 0
ADC1_SE4b	PTC8	ALT 0
ADC1_SE5b	PTC9	ALT 0
ADC1_SE6b	PTC10	ALT 0
ADC1_SE7b	PTC11	ALT 0
ADC0_SE5	PTD1	ALT 0
ADC0_SE6	PTD5	ALT 0
ADC0_SE7	PTD6	ALT 0
ADC1_SE4a	PTE0	ALT 0
ADC1_SE5a	PTE1	ALT 0
ADC1_SE6a	PTE2	ALT 0
ADC1_SE7a	PTE3	ALT 0
ADC0_SE17	PTE24	ALT 0
ADC0_SE18	PTE25	ALT 0

Some ADC inputs have dedicated pins:

ADC0_DP0 / ADC0_DM0

ADC0_DP1 / ADC0_DM1

PGA0_DP / PGA0_DM

ADC0_DP3 / ADC0_DM3

ADC1_DP0 / ADC1_DM0

ADC1_DP1 / ADC1_DM1

PGA1_DP / PGA1_DM

ADC1_DP3 / ADC1_DM3

These are the inputs with differential capabilities and the programmable gain amplifiers.

It is to be noted that the analogue functions are the default functions for the ADC pins.

Conversion time was tested with several setups as reference (K60 running at 100MHz with bus clock at 50MHz):

ADC clock Bus-clock/2 - Pre-scaler /8 - 16 bit mode single-ended - long-sample (24 clocks) – 32x hardware averaging = 460us

Pre-scaler reduced to /1 = 56us

+ Short-Sample mode = 32.4us

+ Bus not divided by 2 = 16us

+ No hardware averaging = 700ns

In 16 bit mode the ADC clock should however be restricted to the range of 2MHz to 12MHz, meaning that not all conversion times can be achieved to specification. Practically, the bus clock needs to be divided by 8 to clock with 6.25MHz, which gives a minimum conversion time of 5.6us – or theoretical maximum sample rate of 178kHz. If the ADC clock can be controlled to be exactly 12MHz the minimum conversion time of 2.6us and maximum sample rate of 383kHz is obtained in 16 bit mode (remembering that this will achieve about 13 bits of accuracy).

The slowest rate using continuous conversion above is equivalent to 2.17kHz.

ADC conversion can be triggered by software, whereby this can be a single conversion or continuous conversion mode.

Hardware triggering is also possible although the source is chip-dependent. The K60 allows the following trigger sources:

- External trigger pin PDBn_EXTRG
- High speed comparators (0, 1 or 2)
- PIT 0, 1, 2 or 3
- FTP0, 1 or 2
- RTC alarm or RTC seconds
- Low power timer

The triggering source is configured in `SIM_SOPT7`.

ADC conversions can be used to trigger DMA transfers.

The K60 has 2 ADC controllers. These both support up to 24 single ended inputs and 4 differential inputs. However only one of the inputs is possible at a time. The two ADC controllers thus allow two inputs to be samples at the same time. If a higher number of inpust are to be sample the sampling must be multiplexed between these inputs under software control.

DAC

The 12 bit Digital to Analogue Converter (DAC) supports DMA operation. The K60 has two DACs and their outputs can drive external pins or be connected internally as reference to the analogue comparator, ADCs or Op-Amp.

The DAC can also be used together with DMA and includes a FIFO especially for this mode of operation.

The 1.2V VREF_OUT or the VDDA voltage can be used as DAC reference. Some degradation of ADC performance is to be expected if VREF_OUT is used by both ADC and DAC at the same time; this is due to the DAC switching. VREF_OUT always need an external load capacitor connected to it spin.

Each DAC has an external trigger input called ADCxSC1A_COCO. This can be also be triggered on ADC conversion completion.

It is to be noted that the comparator modules also contain 6-bit internal DACs. This are internal signals only and are not available on any external pins.

I²C

The K60 has 2 I²C controllers. They are more advanced that the I²C controllers in the M522XX since they support System Management Bus Specification, version 2, have an additional programmable glitch input filter, range I²C address support and can also detected an SCL line which is stuck at '0' for too long.

The I²C controllers can usually be mapped to various pins on the device. The following table shows the possibilities.

I2C0	**I2C0_SDA**	**I2C0_SCL**
I2C0_B_LOW	Port B1 (Alt 2)	Port B0 (Alt 2)
default	Port B3 (Alt 2)	Port B2 (Alt 2)
I2C0_ON_D	Port D9 (Alt 2)	Port D8 (Alt 2)
I2C1	**I2C1_SDA**	**I2C1_SCL**
I2C1_ON_E	Port E0 (Alt 6)	Port E1 (Alt 6)
default	Port C11 (Alt 2)	Port C10 (Alt 2)

SPI

The K60 has 3 SPI interfaces. These are different from the QSPI interface in the Coldfire devices and can also work in slave mode. They are called DSPI because they can also be driven by DMA and have optional 4 deep receive and transmit FIFOs.

Each SPI channel can automatically control up to 6 chip-select lines, which can be de-multiplexed externally to up to 48 lines.

The word length can be between 4 and 16 bits.

The port pin-out possibilities are shown in the following table:

SPI0_SCK	**SPI0_SOUT**	**SPI0_SIN**	**SPI0_CS0**	**SPI0_CS1**	**SPI0_CS2**	**SPI0_CS3**	**SPI0_CS4**	**SPI0_CS5**
Port A15 (Alt 2)	Port A16 (Alt 2)	Port A17 (Alt 2)	Port A14 (Alt 2)	Port C3 (Alt 2)	Port C2 (Alt 2)	Port C1 (Alt 2)	Port C0 (Alt 2)	Port B23 (Alt 3)
Port C5 (Alt 2)	Port C6 (Alt 2)	Port C7 (Alt 2)	Port C4 (Alt 2)	Port D4 (Alt 2)	Port D5 (Alt 2)	Port D6 (Alt 2)		
Port D1 (Alt 2)	Port D2 (Alt 2)	Port D3 (Alt 2)	Port D0 (Alt 2)					
SPI1_SCK	**SPI1_SOUT**	**SPI1_SIN**	**SPI1_CS0**	**SPI1_CS1**	**SPI1_CS2**	**SPI1_CS3**	**SPI1_CS4**	**SPI1_CS5**
Port B11 (Alt 2)	Port B16 (Alt 2)	Port E3 (Alt 2)	Port B10 (Alt 2)	Port B9 (Alt 2)	Port E5 (Alt 2)	Port E6 (Alt 2)		
Port E2 (Alt 2)	Port E1 (Alt 2)	Port B17 (Alt 2)	Port E4 (Alt 2)	Port E0 (Alt 2)				
SPI2_SCK	**SPI2_SOUT**	**SPI2_SIN**	**SPI2_CS0**	**SPI2_CS1**	**SPI2_CS2**	**SPI2_CS3**	**SPI2_CS4**	**SPI2_CS5**
Port B21 (Alt 2)	Port B22 (Alt 2)	Port B23 (Alt 2)	Port B20 (Alt 2)	Port D15 (Alt 2)				
Port D12 (Alt 2)	Port D13 (Alt 2)	Port D14 (Alt 2)	Port D11 (Alt 2)					

The SPI interfaces can be used with or without enabled FIFOs. When the FIFOs are not enabled the SPI operates in simple double-buffered mode. This mode doesn't allow absolute maximum throughput efficiency but is useful when this is not the priority but rather simplicity and compatibility with other code is of more importance. This mode is used when the SD card is operated via SPI (rather than utilising the internal SD controller in the Kinetis K60).

It is to be noted that the configuration of the SPI clock requires careful attention to the parameters than can be set to control the delay between the beginning and end of the frame and the chip select line assertion/negation – also when the chip select lines are not utilised these times influence the beginning or end of the frames by controlling the length of the first or last half bit period, depending on the exact mode of operation. Setting up these delays based on practical measurements is advisable.

SPI FLASH interface: the automatic CS control is used by the µTasker SPI Flash driver due to the fact that it can easily be controlled to assert the CS line (with controllable delay) when the first byte in a stream of data is sent and also negate the CS line automatically (with controllable delay) after the final byte in the stream has been transmitted.

The general configuration consists of programming two registers to define the operating mode and the timing:

```
SPIX_MCR = (SPI_MCR_MSTR | SPI_MCR_DCONF_SPI | SPI_MCR_CLR_RXF |
SPI_MCR_CLR_TXF | SPI_MCR_PCSIS_CS0 | SPI_MCR_PCSIS_CS1 |
SPI_MCR_PCSIS_CS2 | SPI_MCR_PCSIS_CS3 | SPI_MCR_PCSIS_CS4 |
SPI_MCR_PCSIS_CS5);
```

Selects master mode with SCK only generated when transmission is active. The Rx and TX FIFOs are cleared and all CS lines are set to inactive '1'.

```
SPIX_CTAR0 = (SPI_CTAR_DBR | SPI_CTAR_FMSZ_8 | SPI_CTAR_PDT_7 |
SPI_CTAR_BR_2 | SPI_CTAR_CPHA | SPI_CTAR_CPOL); // for 50MHz bus,
25MHz speed and 140ns min de-select time
```

Double bit rate is selected to achieve the maximum operating speed of 25MHz for 50MHz bus and 8 bit words defined. The delay after transfer pre-scaler value is chosen so that the CS line is negated after 140ns delay.

(*SPIX can be replaced by the SPI interface [0, 1 or 2] used*).

The Rx and Tx FIFOs are 4 deep and it is thus possible to prepare up to 4 bytes at a time for transmission. The first or following bytes are transmitted using

```
SPIX_PUSHR = (byte | SPI_PUSHR_CONT | ulChipSelectLine |
SPI_PUSHR_CTAS_CTAR0);
```

This copies the data byte to be transmitted (eg. The SPI Flash command or data) to the transmit FIFO. The chip select mode is set to continuous, so that it is not negated between individual byte transmissions. The chip select to be activated is also defined.

As soon as there is a free place in the transmit FIFOs further bytes of data can be prepared.

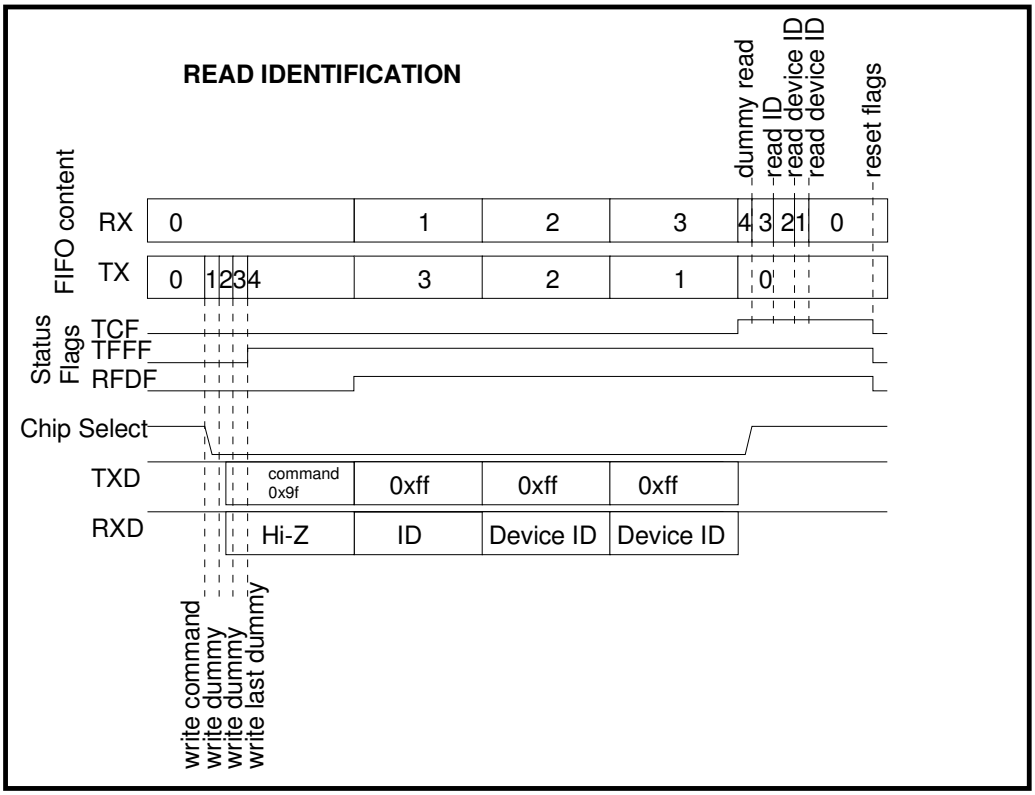
When the final data byte is entered to the output FIFO the negation of the chip select line is also prepared by entering the byte with

```
SPIX_PUSHR = (byte | SPI_PUSHR_EOQ  | ulChipSelectLine |
SPI_PUSHR_CTAS_CTAR0);
```

`SPI_PUSHR_EOQ` signals that the FIFO entry is the final byte in the transfer sequence so the chip select is negated after the data has been sent.

For every byte transmitted one reception byte enters the receive FIFO. It is therefore important to know how many bytes have been sent before reception data is returned so that these can be appropriately emptied beforehand.

The following diagrams show typical data transfers, and serve as examples of the operation of the drivers:



Read of SPI Flash identification: this is a four byte transfer whereby it is possible to write four bytes of data (the command followed by three dummy bytes) immediately. The diagram shows this technique (the actual method used in the drivers may depart from this exact technique and not keep the transmit FIFO completely full but the principle remains generally valid) in operation since three bytes are immediately written to the transmit FIFO, followed by a terminating byte, which configures the negation of the chip select line after its complete transmission.

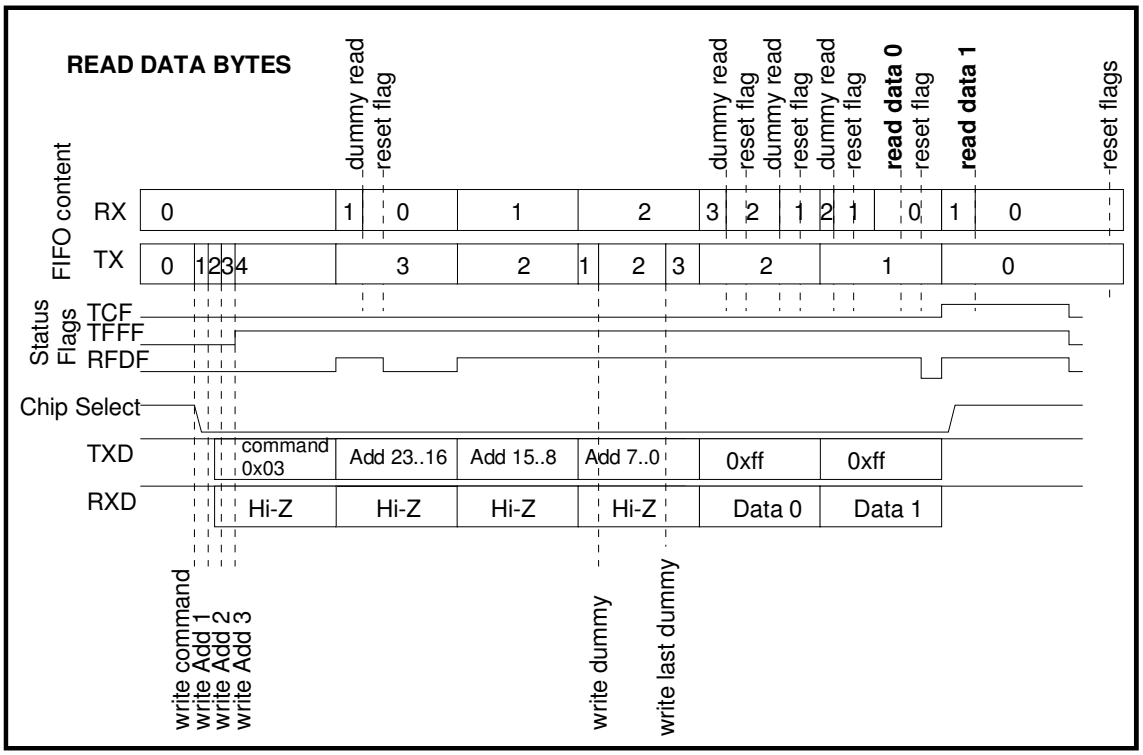
Various status flags are shown during the frame transfer. All flags are assumed to be '0' before the transfer begins, which is achieved by actively resetting then in the status register. *It is important to note that the flags are not self-clearing and so the driver needs to manually reset each once its condition has been handled so that the next flag state change can take place again.*

For simplicity it is in fact possible to use only the receive flag (RFDF) for all control due to the fact that there is always a reception byte with each transmitted byte. This flag indicates that there is at least one byte of data in the receive FIFO which can be read.

The diagram shows also the number of data bytes in the transmit and receive FIFOs, whereby the driver attempts mainly to keep the transmitter FIFO as full as possible and to read received bytes as quickly as possible. It would be generally possible to always transmit and receive small chunks of 4 bytes but this may result in short dead-times between the chunks, which the technique of keeping the transmit buffer 'fed' avoids (with possible exceptions when interrupts can temporarily suspend the process so that the transmit FIFO completely empties).

Note that the RFDR flag is set once the first reception byte is received (thus actually takes place about 2 clocks after the byte has been completely transferred on the SPI bus) and so the driver may already start processing the reception byte at that instance. In the diagram the

processing of all 4 reception bytes takes place at the end of the frame, but this is mainly to show the operation of the FIFOs. The first received byte is a dummy byte since the SPI Flash doesn't respond during the transmission of the command byte (the line is in fact in the high-impedance state).



The second diagram shows a typical read case where three address bytes follow the command. Again the FIFO operation is illustrated. For each data byte to be read (only two are read in this diagram) the driver sends a dummy byte with value 0xff, which is transmitted and at the same time clocks in the data from the SPI Flash to the receiver FIFO. In this case there are also more data bytes in the transfer than the FIFO has space for and the driver operation, controlled by the RFDF flag, is more visible. In the illustration the first read of the FIFO takes place quickly after the RFDF flag has been set, after which the flag is manually cleared. Then there is a longer interval between the flag being set and the data being extracted – this is for illustration purposes since the FIFO operation allows the transmitter and receiver to continue operation as long as the transmit FIFO has been adequately "fed" and there is less impact from processor delays up the a point when the transmit FIFO is completely sent.

SDHC

The K60 has a powerful SDHC (Secured Digital Host Controller). This is of use when working with SD cards to improve transfer efficiency.

The following signals are involved for interfacing with SD cards:

SDHC_DCLK	PTE2	Output	Clock signal	
SDHC_CMD	PTE3	I/O	Sends commands and receives responses	
SDHC_D0	PTE1	I/O	DAT0	
SDHC_D1	PTE0	I/O	DAT1	
SDHC_D2	PTE5	I/O	DAT2	
SDHC_D3	PTE4	I/O	DAT3	
CARD DETECT	PTE28			
WRITE PROTECT	PTE27			

In order for the controller driver interface to be compatible with the µTasker utFAT module it needs to support the following hardware specific routines:

```
extern void fnInitSDCardInterface(void);

extern int  fnSendSD_command(const unsigned char ucCommand[6],
unsigned char *ucResult, unsigned char *ptrReturnData);

extern int  fnGetSector(unsigned char *ptrBuf);

extern int  fnReadPartialSector(unsigned char *ptrBuf, unsigned
short usStart, unsigned short usStop);

extern int  fnPutSector(unsigned char *ptrBuf);
```

FlexCAN

The Kinetis K60 has two FlexCAN channels which are very similar to the single FlexCAN controller in the M52235 and other Coldfire devices. The pin-out of the CAN_TX and CAN_RX lines are shown in the following table:

Channel	CAN_TX	CAN_RX
0 - default	PTA12 – Alt 2	PTA13 – Alt 2
0 – `define CAN0_ON_PB`	PTB18 – Alt 2	PTB19 – Alt 2
1 - default	PTC17 – Alt 2	PTC16 – Alt 2
0 – `define CAN1_ON_PE`	PTE14 – Alt 2	PTE25 – Alt 2

The µTasker CAN driver supports both of the channels as does the µTasker simulator, which can work with the KOMODO CAN DUO for connection of the simulator to up to two CAN buses and also monitor their operation at the same time. See the video: http://www.youtube.com/watch?v=Ha8cv_XEvco

User's guide: http://www.utasker.com/docs/uTasker/uTaskerCAN.PDF

FlexTimer (PWM)

The Kinetis has up to 4 FlexTimers, which are each based on their own 16 bit counters. Each FlexTimer has up to 8 channels, which all have their own port for input capture or output compare. They can be used to generate either edge-aligned or centre-aligned PWM outputs.

Not all FlexTimer and channels are available in all devices. For example, the K60 has FlexTimers 0, 1 and 2, whereby FlexTimer 0 has all 8 channels available and FlexTimers 1 and 2 have only channels 0 and 1 available.

Channel	FTM0
0 – default 0 – FTM0_0_ON_C	PTA3 – Alt 3 PTC1 – Alt 4
1 – default 1 – FTM0_1_ON_C	PTA4 – Alt 3 PTC2 – Alt 4
2 – default 2 – FTM0_2_ON_C	PTA5 – Alt 3 PTC3 – Alt 4
3 – default 3 – FTM0_3_ON_C	PTA6 – Alt 3 PTC4 – Alt 4
4 – default 4 – FTM0_4_ON_D	PTA7 – Alt 3 PTD4 – Alt 4
5 – default 5 – FTM0_5_ON_D	PTA0 – Alt 3 PTD5 – Alt 4
6 – default 6 – FTM0_6_ON_D	PTA1 – Alt 3 PTD6 – Alt 4
7 – default 7 – FTM0_7_ON_D	PTA2 – Alt 3 PTD7 – Alt 4

Channel	FTM1
0 – default 0 – FTM1_0_ALT_A 0 – FTM1_0_ON_B	PTA8 – Alt 3 PTA12 – Alt 3 PTB0 – Alt 3
1 – default 1 – FTM1_1_ALT_A 1 – FTM1_1_ON_B	PTA9 – Alt 3 PTA13 – Alt 3 PTB1 – Alt 3

Channel	FTM2
0 – default 0 – FTM2_0_ON_B	PTA10 – Alt 3 PTB18 – Alt 3
1 – default 1 – FTM2_1_ON_B	PTA11 – Alt 3 PTB19 – Alt 3

- Although the PWM operation can be edge or centred-aligned the setting is shared by all outputs on a single FlexTimer since the operation of its counter is set to either up counting mode (edge-alignment) or up/down counting mode (centre-alignment).
- The FlexTimer can be clocked from the system clock, the fixed clock MCGFFCLK or from an external clock. There are two possible FlexTimer external clock pins which can be selected for use by any FlexTimer module, on PA18 (default) or PA19 (FTM_CLKIN_1). These are however multiplexed with EXTAL and XTAL and so need to be used carefully.
- The behaviour of the Flex Timer counter and its outputs during debugging (BDM mode) can be defined by the selection of the define `FTM_DEBUG_BEHAVIOUR` in `app_hw_kinetis.h`. It can be allowed to run or stopped and its outputs can be frozen, continue running or be held in a defined state.

NAND Flash Controller

Some of the high speed parts such as the K70 include a NAND Flash controller.

This can be clocked from OSC0ERCLK or from a pre-scaled version of the bus clock, PLL or OSC0ERCLK.

It can interface gluelessly to standard 8 and 16 bit NAND Flash devices with page sizes of 512 bytes, 2k, 4k and 8k.

It has a 9k RAM buffer, supports ECC and BCH (ECC with multi-bit error correction).

The TWR-K70F120M board has a MT29F2G16ABAEAWP on it. This is a 2Gbit NAND Flash from Micron in a 48TSSOP housing. It is connected in 16 bit mode and has a total of 256Mbytes data space.

K40 Notes:

The K40 in 144 pin housing is highly compatible to the K60. The K40 has some extra pin functions such as FlexBus option on Port E and Port A.
Port B has SLCD interface – some FlexBus pins have been removed from Ports B and Port C to Port E.
Port bits D8 and D9 are not connected on the K40
Some Port C bits are at different pin locations on the LQFP package.
K40 doesn't have Ethernet.

The segment LCD interface consists of 40 lines - LCD_P0 .. LCD_P39

- KwikStik SLCD has 306 segments
- Tower board adapter has 27 segments

SLCD controller support has been added to the project as detailed in the document: http://www.utasker.com/docs/uTasker/uTasker_SLCD.pdf and shown in the video: http://www.youtube.com/watch?v=nm2DmZv1rj8

Kinetis Device Family

The µTasker port simulator supports all devices (K10, K20, K30, K40, K50, K51, K52, K53, K60, K61 and K70) in 80/81, 100, 121, 144, 196 or 256 pin packages as appropriate.