SPI Driver Specification

This note describes some of the features and specifications for the SPIDriver module as at 20160316. This is revision 0.01, JWB
It reflects a work in progress.
Achievements to date: Based on the iMx6 HAL the following is functional:
-Reading SPI Chip capabilities
-Configuring the Chip
-Reading a set of available GPIO pins that can be used with an external decoder to expand the SPI device address range
-Selecting a subset of available pins for use as I/O decode pins and mapping them to a specific SPI channel
-Initiating and completing SPI data transfers in foreground polled mode. Other modes are specified in the header file and structures, and will be implemented in due course.

The SPIDriver module is written in C and uses the capabilities of the SPI HALDevice exported by the HAL, or simulated in another module.

The following SWIs exist:

&59880	SPID_Version
&59881	SPID_Initialise
&59882	SPID_Control
&59883	SPID_Transfer
&59884	SPID_Status

The SPID_Version and SPID_Status SWIs are in place but currently unused.

SPID_Version:
Currently returns 0

SPID_Initialise:
On entry, reason code in R0:
0: ListDevices:- returns a pointer to a spidevice_list structure detailing the SPI hosts present.
1: ReadCapabilities:- given a spidevice pointer in R1, it returns a pointer to a spi_capabilites structure for this device.
2: SetConfiguration:- configures a spidevice (pointer in R1) to the the spi_settings (pointer in R2).
3: PinMap:- for spidevice (pointer in R1) sub reason (in R2)
    0:- return in R0 a pointer to a null terminated list of spi_pinmaplist structures. Provides details of available GPIO pins that could be allocated for SPI device decode
    1:- map pinspec in R3 to usage in R4 (bit position 3..0 for SPI ChipSelect pin number in bits 7..4)
    2:- set decode pins to the value in R3 Bits 7..4 = ChipSelect, 3..0 value
    2:- Clear all pin mapping and disable pins if R3 = 44 and R4 = -44

SPID_Control:
Currently returns 0

SPID_Transfer:
Initiate a SPI transfer using the spi_transfer pointer provided in R0. Currently only polled transfer is implemented. The spi_transfer structure typically contains a pointer to the spi_settings structure used in the SPID_Initialise - SetConfiguration call above. It is modified as needed to control this particular transfer.

SPID_Status:
Currently returns 0

There is one * command, SPIDevices which lists the SPI controllers known to the SPIDriver

The following listing is taken from the Driver sources as at 20160316 and defines all the structures used:

```
/* This source code in this file is licensed to You by Castle Technology
 * Limited ("Castle") and its licensors on contractual terms and conditions
 * ("Licence") which entitle you freely to modify and/or to distribute this
 * source code subject to Your compliance with the terms of the Licence.
 *
 * This source code has been made available to You without any warranties
 * whatsoever. Consequently, Your use, modification and distribution of this
 * source code is entirely at Your own risk and neither Castle, its licensors
 * nor any other person who has contributed to this source code shall be
 * liable to You for any loss or damage which You may suffer as a result of
 * Your use, modification or distribution of this source code.
 *
 * Full details of Your rights and obligations are set out in the Licence.
 * You should have received a copy of the Licence with this source code file.
 * If You have not received a copy, the text of the Licence is available
 * online at www.castle-technology.co.uk/riscosbaselicence.htm
 */
#ifndef spidriver_H
#define spidriver_H

/* keep this in sync with the SPIDevice definition in kernel.hdr */
typedef struct spi_device
{
  struct device    dev;          // normal HAL Device structure
  void           (*hal_cap)(struct device *, void *); //struct spi_capabilities*
  void           (*hal_config)(struct device *, void*); // struct spi_settings*
  void           (*hal_transfer)(struct device *, void*);// struct spi_transfer*
  unsigned       (*hal_pinmap)(struct device *, int,int,int);  //
  void           (*hal_dev3)(struct device *);
  void           (*hal_dev4)(struct device *);
  void           (*hal_dev5)(struct device *);
} spidevice;
// structure supplied by driver, used to report the SPIdevice capabilities
// ownership kept by the Driver. This is RMA
typedef struct spi_capabilities
{
  spidevice *spidev;
  unsigned int fastclock;          // clock Hz
  unsigned int slowclock;          // slow delay clock Hz or null
  unsigned int maxburstbitlength;  // burst length in bits
//
  unsigned int maxhtbits;          // HT burst length in bits
  unsigned int rxfifosize;         // bytes
  unsigned int txfifosize;         // bytes
  unsigned int msgdatafifosize;    // bytes
```

```
//
  unsigned int maxdelaySS2CLK;     // max SPI clock delay from SS to first SCLK
  unsigned int maxSPDelay;         // max delay between SPI samples
  void *        wrfifoaddr;        // (from HAL) fifo write address if PolledIO
  void *        rdfifoaddr;        // (from HAL) fifo read address if PolledIO
//
  void *        wrpolladdr;        // (from HAL) wr data poll bit address
  unsigned int  wrpollmask;        // (from HAL)1 bit set, then bit is active hi
                                   // (from HAL)1 bit clr then bit is active low
  void *        rdpolladdr;        // (from HAL) read data poll bit address
  unsigned int  rdpollmask;        // (from HAL)1 bit set, then bit is active hi
                                   // (from HAL)1 bit clr then bit is active low
// all below are unsigned char. must be groups of 4 for word alignment
  unsigned char datawidth;         // data register byte width (1,2,4,etc)
  unsigned char spiclockprediv;    // predivide max value . 0 = divide by 1
  unsigned char spiclockpostdiv;   // divide max value.  2^n,0 = divide by 1
                                   // n = divide by 2^n
  unsigned char mastermode;        // bitmap (0..7) - channels with master mode
//
  unsigned char slavemode;         // bitmap (0..7) - channels with slave mode
  unsigned char modebits;          // bit 0 set- Hardware trigger (HT)available
                                   //     1 set- fifo write starts SPI burst
                                   //     2 set- software trigger (XCH)available
                                   //     3 set- can use slowclock for SPDelay
  unsigned char SCLKrest;          // bit set if inactive level can be altered
  unsigned char SDATArest;         // bit set if inactive level can be altered
//
  unsigned char SSpol;             // bit set if SS active level can be altered
  unsigned char SSctl;             // bit set if SS active mode can be altered
  unsigned char SCLKpol;           // bit set if SCLK polarity can be altered
  unsigned char SSCLKphase;        // bit set if SCLK phase can be altered
//
//  unsigned char pad1;
//  unsigned char pad2;
//  unsigned char pad3;
} spi_capabilities;

/* linked list structure for devices found */
typedef struct spi_devicelist
{
  spidevice *spidev;
  struct spi_devicelist *next;
  struct spi_capabilities cap;  // filled in by querying the HAL
} spi_devicelist;

// structure sent to SPIdevice to configure it- needs to be in RMA. Ownership
//  remains with sender
typedef struct spi_settings
{
  spidevice *dev;
  unsigned int clock;              // SPI clock in Hz
  unsigned int delayclock;         // SPI clock periods between transfers
  unsigned int burstlength;        //  burst length in bits
                                   //  if the burst length is not an exact multiple
                                   //  of the spi_capabilities.datawidth, the
                                   //  fractional part occupies the bottom bits
                                   //  of the first data read or write
                                   //  e.g. datawidth 4 bytes = 32 bits
                                   //       burst length 50 bits.
                                   //       first read or write word contains
                                   //            first 18 (50-32) bits
                                   //       remaining transfers are all 32 bits wide
//  all below are unsigned char, in groups of 4, padded up
  unsigned char drcontrol;         // 0x0 = dont care
                                   // 0x1 = host burst triggers on falling
                                   //       edge of SPI_RDY line
                                   // 0x2 = host burst triggers on rising edge
                                   //
  unsigned char mode;              // bit field, per SS, 1 - master mode
  unsigned char modebits;          // bit 0 set - (HT) Hardware trigger
                                   //     1 clr - software bit starts SPI burst
                                   //     1 set - fifo write starts SPI burst
                                   //     2 set - (XCH) software trigger burst
                                   //     3 set - use slowclock for SPDelay
                                   // bit 4 set - these settings are already set
                                   //             (written by HAL on first call)
                                   //             (client writes 0 on change)
  unsigned char SCLKlevel;         // per SS, inactive SCLK level
// next word
  unsigned char SDATAlevel;        // per SS, inactive SDATA level
  unsigned char SSlevel;           // per SS, active SS level
  unsigned char SPIburst;          // per SS, 1 - multi SPI bursts, 0 - single
  unsigned char SCLKpol;           // per SS, 1 - active HI, idle at 0, etc
// next word
  unsigned char SCLKphase;         // per SS, 0 - phase 0, 1 - phase 1
                                   //
                                   //

 unsigned char pad1;               // pad to word align
 unsigned char pad2;               // pad to word align
 unsigned char pad3;               // pad to word align

} spi_settings;

// structure used to initialise a transfer
typedef struct spi_transfer
{
  spi_settings  *spisettings;      // pointer->associated spi settings structure
  unsigned char * wr_addr;         // Write buffer address
  unsigned char * rd_addr;         // buffer to put receive data in, or NULL
  void *        cb_addr;           // background done callback address,or NULL
  unsigned int  burstlength;       //  burst length in bits
                                   //  if the burst length is not an exact multiple
                                   //  of the spi_capabilities.datawidth, the
                                   //  fractional part occupies the bottom bits
                                   //  of the first data read or write
                                   //  e.g. datawidth 4 bytes = 32 bits
                                   //       burst length 50 bits.
                                   //       first read or write word contains
                                   //            first 18 (50-32) bits
                                   //       remaining transfers are all 32 bits wide
  unsigned int timeout;            // transfer timeout in centiseconds
                                   //  all below are unsigned char, in groups of 4, padded up
  unsigned char dma_mode;          // bit 0: use PolledIO if 0, else dma if 1
                                   //     1: use background transfer
  unsigned char channel;           // spi channel to use
                                   // bits 3..0 i/o decode for this chip select
                                   // bits 7..4 chip select  (0 to max there

  unsigned char pad1;               // pad to word align
  unsigned char pad2;               // pad to word align
//  unsigned char pad3;               // pad to word align
} spi_transfer;

// structure used to report and specify gpio pin available for mapping
// for external hardware chip select decoding
// the pinspec value is used to select this pin for assignment
// using the
typedef struct spi_pinmaplist
{
  int           pinspec;            // pin# + (gpio<<5) + (state<<8)
  int           internal1;          // internal use by HAL
  int           internal2;          // ditto
  int           internal3;          // ditto
  char          pinname[12];        // null terminated text name
} spi_pinmaplist;
typedef enum spi_pinmapaction {list,map,select,clear} spi_pinmapaction_t;
typedef struct spi_pinmap
{
  spi_pinmapaction_t action;
} spi_pinmap;

#endif
/* Hdr2H exported hdr file appended below in exported version */
```