```c
/* SAM4S4B_pio.h
 *
 * cferrarin@g.hmc.edu
 * kpezeshki@g.hmc.edu
 * 12/11/2018
 *
 * Contains base address locations, register structs, definitions, and
  functions for the PIO
 * (Parallel Input/Output Controller) peripheral of the SAM4S4B
  microcontroller. */

#ifndef SAM4S4B_PIO_H
#define SAM4S4B_PIO_H

#include <stdint.h>

////////////////////////////////////////////////////////////////////////////
 ////////////////////
// PIO Base Address Definitions
////////////////////////////////////////////////////////////////////////////
 ////////////////////

#define PIOA_BASE  (0x400E0E00U) // PIOA Base Address
#define PIOB_BASE  (0x400E1000U) // PIOB Base Address


////////////////////////////////////////////////////////////////////////////
 ////////////////////
// PIO Registers
////////////////////////////////////////////////////////////////////////////
 ////////////////////

// Peripheral struct for a PIO peripheral (either PIOA or PIOB)
typedef struct {
    volatile uint32_t PIO_PER;        // (Pio Offset: 0x0000) PIO Enable
     Register
    volatile uint32_t PIO_PDR;        // (Pio Offset: 0x0004) PIO Disable
     Register
    volatile uint32_t PIO_PSR;        // (Pio Offset: 0x0008) PIO Status
     Register
    volatile uint32_t Reserved1[1];
    volatile uint32_t PIO_OER;        // (Pio Offset: 0x0010) Output Enable
     Register
    volatile uint32_t PIO_ODR;        // (Pio Offset: 0x0014) Output Disable
     Register
    volatile uint32_t PIO_OSR;        // (Pio Offset: 0x0018) Output Status
     Register
    volatile uint32_t Reserved2[1];
    volatile uint32_t PIO_IFER;       // (Pio Offset: 0x0020) Glitch Input
     Filter Enable Register
```

```
volatile uint32_t PIO_IFDR;       // (Pio Offset: 0x0024) Glitch Input
 Filter Disable Register
volatile uint32_t PIO_IFSR;       // (Pio Offset: 0x0028) Glitch Input
 Filter Status Register
volatile uint32_t Reserved3[1];
volatile uint32_t PIO_SODR;       // (Pio Offset: 0x0030) Set Output Data
 Register
volatile uint32_t PIO_CODR;       // (Pio Offset: 0x0034) Clear Output Data
 Register
volatile uint32_t PIO_ODSR;       // (Pio Offset: 0x0038) Output Data
 Status Register
volatile uint32_t PIO_PDSR;       // (Pio Offset: 0x003C) Pin Data Status
 Register
volatile uint32_t PIO_IER;        // (Pio Offset: 0x0040) Interrupt Enable
 Register
volatile uint32_t PIO_IDR;        // (Pio Offset: 0x0044) Interrupt Disable
 Register
volatile uint32_t PIO_IMR;        // (Pio Offset: 0x0048) Interrupt Mask
 Register
volatile uint32_t PIO_ISR;        // (Pio Offset: 0x004C) Interrupt Status
 Register
volatile uint32_t PIO_MDER;       // (Pio Offset: 0x0050) Multi-driver
 Enable Register
volatile uint32_t PIO_MDDR;       // (Pio Offset: 0x0054) Multi-driver
 Disable Register
volatile uint32_t PIO_MDSR;       // (Pio Offset: 0x0058) Multi-driver
 Status Register
volatile uint32_t Reserved4[1];
volatile uint32_t PIO_PUDR;       // (Pio Offset: 0x0060) Pull-up Disable
 Register
volatile uint32_t PIO_PUER;       // (Pio Offset: 0x0064) Pull-up Enable
 Register
volatile uint32_t PIO_PUSR;       // (Pio Offset: 0x0068) Pad Pull-up
 Status Register
volatile uint32_t Reserved5[1];
volatile uint32_t PIO_ABCDSR1;  // (Pio Offset: 0x0070) Peripheral Select
 Register 1
volatile uint32_t PIO_ABCDSR2;  // (Pio Offset: 0x0074) Peripheral Select
 Register 2
volatile uint32_t Reserved6[2];
volatile uint32_t PIO_IFSCDR;     // (Pio Offset: 0x0080) Input Filter Slow
 Clock Disable Register
volatile uint32_t PIO_IFSCER;     // (Pio Offset: 0x0084) Input Filter Slow
 Clock Enable Register
volatile uint32_t PIO_IFSCSR;     // (Pio Offset: 0x0088) Input Filter Slow
 Clock Status Register
volatile uint32_t PIO_SCDR;       // (Pio Offset: 0x008C) Slow Clock
 Divider Debouncing Register
volatile uint32_t PIO_PPDDR;      // (Pio Offset: 0x0090) Pad Pull-down
 Disable Register
```

```
volatile uint32_t PIO_PPDER;      // (Pio Offset: 0x0094) Pad Pull-down
 Enable Register
volatile uint32_t PIO_PPDSR;      // (Pio Offset: 0x0098) Pad Pull-down
 Status Register
volatile uint32_t Reserved7[1];
volatile uint32_t PIO_OWER;       // (Pio Offset: 0x00A0) Output Write
 Enable
volatile uint32_t PIO_OWDR;       // (Pio Offset: 0x00A4) Output Write
 Disable
volatile uint32_t PIO_OWSR;       // (Pio Offset: 0x00A8) Output Write
 Status Register
volatile uint32_t Reserved8[1];
volatile uint32_t PIO_AIMER;      // (Pio Offset: 0x00B0) Additional
 Interrupt Modes Enable Register
volatile uint32_t PIO_AIMDR;      // (Pio Offset: 0x00B4) Additional
 Interrupt Modes Disables Register
volatile uint32_t PIO_AIMMR;      // (Pio Offset: 0x00B8) Additional
 Interrupt Modes Mask Register
volatile uint32_t Reserved9[1];
volatile uint32_t PIO_ESR;        // (Pio Offset: 0x00C0) Edge Select
 Register
volatile uint32_t PIO_LSR;        // (Pio Offset: 0x00C4) Level Select
 Register
volatile uint32_t PIO_ELSR;       // (Pio Offset: 0x00C8) Edge/Level Status
 Register
volatile uint32_t Reserved10[1];
volatile uint32_t PIO_FELLSR;     // (Pio Offset: 0x00D0) Falling Edge/Low
 Level Select Register
volatile uint32_t PIO_REHLSR;     // (Pio Offset: 0x00D4) Rising Edge/ High
 Level Select Register
volatile uint32_t PIO_FRLHSR;     // (Pio Offset: 0x00D8) Fall/Rise - Low/
 High Status Register
volatile uint32_t Reserved11[1];
volatile uint32_t PIO_LOCKSR;     // (Pio Offset: 0x00E0) Lock Status
volatile uint32_t PIO_WPMR;       // (Pio Offset: 0x00E4) Write Protect
 Mode Register
volatile uint32_t PIO_WPSR;       // (Pio Offset: 0x00E8) Write Protect
 Status Register
volatile uint32_t Reserved12[5];
volatile uint32_t PIO_SCHMITT;    // (Pio Offset: 0x0100) Schmitt Trigger
 Register
volatile uint32_t Reserved13[19];
volatile uint32_t PIO_PCMR;       // (Pio Offset: 0x150) Parallel Capture
 Mode Register
volatile uint32_t PIO_PCIER;      // (Pio Offset: 0x154) Parallel Capture
 Interrupt Enable Register
volatile uint32_t PIO_PCIDR;      // (Pio Offset: 0x158) Parallel Capture
 Interrupt Disable Register
volatile uint32_t PIO_PCIMR;      // (Pio Offset: 0x15C) Parallel Capture
 Interrupt Mask Register
```

```
    volatile uint32_t PIO_PCISR;      // (Pio Offset: 0x160) Parallel Capture
     Interrupt Status Register
    volatile uint32_t PIO_PCRHR;      // (Pio Offset: 0x164) Parallel Capture
     Reception Holding Register
    volatile uint32_t PIO_RPR;        // (Pio Offset: 0x168) Receive Pointer
     Register
    volatile uint32_t PIO_RCR;        // (Pio Offset: 0x16C) Receive Counter
     Register
    volatile uint32_t Reserved14[2];
    volatile uint32_t PIO_RNPR;       // (Pio Offset: 0x178) Receive Next
     Pointer Register
    volatile uint32_t PIO_RNCR;       // (Pio Offset: 0x17C) Receive Next
     Counter Register
    volatile uint32_t Reserved15[2];
    volatile uint32_t PIO_PTCR;       // (Pio Offset: 0x188) Transfer Control
     Register
    volatile uint32_t PIO_PTSR;       // (Pio Offset: 0x18C) Transfer Status
     Register
} Pio;

// Pointers to Pio-sized chunks of memory at each PIO peripheral
#define PIOA ((Pio*) PIOA_BASE)
#define PIOB ((Pio*) PIOB_BASE)


////////////////////////////////////////////////////////////////////////////////
 ////////////////////
// PIO Definitions
////////////////////////////////////////////////////////////////////////////////
 ////////////////////

// Values which "val" can take on in pioWritePin()
#define PIO_LOW  0 // Value to write a pin low (0 V)
#define PIO_HIGH 1 // Value to write a pin high (3.3 V)

// Arbitrary port IDs used to easily find a pin's port
#define PIO_PORT_ID_A 0 // Arbitrary ID for PIO Port A
#define PIO_PORT_ID_B 1 // Arbitrary ID for PIO Port B

// Values which "function" can take on in pioPinMode()
#define PIO_INPUT     0 // Arbitrary ID for an input I/O line
#define PIO_OUTPUT    1 // Arbitrary ID for an output I/O line
#define PIO_PERIPH_A  2 // Arbitrary ID for peripheral function A
#define PIO_PERIPH_B  3 // Arbitrary ID for peripheral function B
#define PIO_PERIPH_C  4 // Arbitrary ID for peripheral function C
#define PIO_PERIPH_D  5 // Arbitrary ID for peripheral function D
#define PIO_PULL_DOWN 6 // Arbitrary ID for a pull-down resistor
#define PIO_FLOATING  7 // Arbitrary ID for neither a pull-up nor a pull-down
 resistor
```

```
// Pin definitions for every PIO pin, which "pin" can take on in several
 functions
#define PIO_PA0  0
#define PIO_PA1  1
#define PIO_PA2  2
#define PIO_PA3  3
#define PIO_PA4  4
#define PIO_PA5  5
#define PIO_PA6  6
#define PIO_PA7  7
#define PIO_PA8  8
#define PIO_PA9  9
#define PIO_PA10 10
#define PIO_PA11 11
#define PIO_PA12 12
#define PIO_PA13 13
#define PIO_PA14 14
#define PIO_PA15 15
#define PIO_PA16 16
#define PIO_PA17 17
#define PIO_PA18 18
#define PIO_PA19 19
#define PIO_PA20 20
#define PIO_PA21 21
#define PIO_PA22 22
#define PIO_PA23 23
#define PIO_PA24 24
#define PIO_PA25 25
#define PIO_PA26 26
#define PIO_PA27 27
#define PIO_PA28 28
#define PIO_PA29 29
#define PIO_PA30 30
#define PIO_PA31 31
#define PIO_PB0  32
#define PIO_PB1  33
#define PIO_PB2  34
#define PIO_PB3  35
#define PIO_PB4  36
#define PIO_PB5  37
#define PIO_PB6  38
#define PIO_PB7  39
#define PIO_PB8  40
#define PIO_PB9  41
#define PIO_PB10 42
#define PIO_PB11 43
#define PIO_PB12 44
#define PIO_PB13 45
#define PIO_PB14 46
```

```
// Writing any other value in this field aborts the write operation of the WPEN
 bit.
// Always reads as 0.
#define PIO_WPMR_WPKEY_PASSWD (0x50494Fu << 8)


////////////////////////////////////////////////////////////////////////////////
 ////////////////////
// PIO Functions
////////////////////////////////////////////////////////////////////////////////
 ////////////////////

/* Initializes the PIO peripheral by enabling the Master Clock to PIOA and
 PIOB. */
void pioInit() {
    pmcEnablePeriph(PMC_ID_PIOA);
    pmcEnablePeriph(PMC_ID_PIOB);
}

/* Returns the port ID that corresponds to a given pin.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- return: a PIO port ID, e.g. PIO_PORT_ID_A */
int pioPinToPort(int pin) {
    return pin >> 5;
}

/* Returns a pointer to the given port's base address.
 *    -- port: a PIO port ID, e.g. PIO_PORT_ID_A
 *    -- return: a pointer to a Pio-sized block of memory at the port "port" */
Pio* pioPortToBase(int port) {
    return port ? PIOB : PIOA;
}

/* Given a pin, returns a pointer to the corresponding port's base address.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- return: a pointer to a Pio-sized block of memory at the pin's port */
Pio* pioPinToBase(int pin) {
    return pioPortToBase(pioPinToPort(pin));
}

/* Sets a function (either I/O behavior, peripheral, or setting) of a pin.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- function: a PIO function ID, e.g. PIO_PERIPH_C. While I/O functions
  (PIO_INPUT, PIO_OUTPUT)
 *       and peripherals (PIO_PERIPH_A - PIO_PERIPH_D) are mutually exclusive,
  settings
 *       (PIO_PULL_DOWN, PIO_FLOATING), can always be altered.
 * Note: upon reset, pins are configured as input I/O lines (as opposed to
  peripheral functions),
 * the peripheral defaults to PIO_PERIPH_A, the pull-up resistor is enabled,
  and the pull-down
```

```
 * resistor is disabled. All other optional pin functions, which are not
  provided in this driver,
 * are disabled upon reset. Note also that pin PA31 is used for the FPGA clock,
  and should not be
 * altered */
void pioPinMode(int pin, int function) {
    Pio* port = pioPinToBase(pin);
    int offset = pin % 32;

    switch (function) {
        case PIO_INPUT:
            break; // Do nothing, since this is default behavior
        case PIO_OUTPUT:
            port->PIO_OER     |=  (1 << offset); // Configures an I/O line as
             an output
            break;
        case PIO_PERIPH_A:
            port->PIO_PDR     |=  (1 << offset); // Sets a pin to be
             peripheral-controlled
            port->PIO_ABCDSR1 &= ~(1 << offset); // Sets the peripheral which
             controls a pin
            port->PIO_ABCDSR2 &= ~(1 << offset); // Sets the peripheral which
             controls a pin
            break;
        case PIO_PERIPH_B:
            port->PIO_PDR     |=  (1 << offset); // Sets a pin to be
             peripheral-controlled
            port->PIO_ABCDSR1 |=  (1 << offset); // Sets the peripheral which
             controls a pin
            port->PIO_ABCDSR2 &= ~(1 << offset); // Sets the peripheral which
             controls a pin
            break;
        case PIO_PERIPH_C:
            port->PIO_PDR     |=  (1 << offset); // Sets a pin to be
             peripheral-controlled
            port->PIO_ABCDSR1 &= ~(1 << offset); // Sets the peripheral which
             controls a pin
            port->PIO_ABCDSR2 |=  (1 << offset); // Sets the peripheral which
             controls a pin
            break;
        case PIO_PERIPH_D:
            port->PIO_PDR     |=  (1 << offset); // Sets a pin to be
             peripheral-controlled
            port->PIO_ABCDSR1 |=  (1 << offset); // Sets the peripheral which
             controls a pin
            port->PIO_ABCDSR2 |=  (1 << offset); // Sets the peripheral which
             controls a pin
            break;
        case PIO_PULL_DOWN:
            port->PIO_PUDR    |=  (1 << offset); // Disables the pull-up
             resistor
```

```
            port->PIO_PPDER   |=  (1 << offset); // Enables the pull-down
             resistor
        case PIO_FLOATING:
            port->PIO_PUDR    |=  (1 << offset); // Disables the pull-down
             resistor
    }
}

/* Reads the digital voltage on a pin configured as an input I/O line.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- return: a PIO value ID, either PIO_HIGH or PIO_LOW */
int pioReadPin(int pin) {
    Pio* port = pioPinToBase(pin);
    int offset = pin % 32;
    return ((port->PIO_PDSR) >> offset) & 1;
}

/* Writes a digital voltage to a pin configured as an output I/O line.
 *    -- pin: a PIO pin ID, e.g. PIO_PA3
 *    -- val: a PIO value ID, either PIO_HIGH or PIO_LOW */
void pioWritePin(int pin, int val) {
    Pio* port = pioPinToBase(pin);
    int offset = pin % 32;
    if (val) {
        port->PIO_SODR |= (1 << offset);
    } else {
        port->PIO_CODR |= (1 << offset);
    }
}

/* Switches the digital voltage on a pin configured as in output I/O line
 *    -- pin: a PIO pin ID, e.g. PIO_PA3 */
void pioTogglePin(int pin) {
    int currentVal = pioReadPin(pin);
    pioWritePin(pin, !currentVal);
}


#endif
```