

Chapter 14: Working Directly with the Control Registers

tw rev. 15.4.2024

If you use or reference these slides or the associated textbook, please cite the original authors' work as follows:

Wilmshurst, T., Spink, T. & Toulson, R. (2024). Fast and Effective Embedded Systems Design – From Bits and Bytes to IoT, with the ARM Mbed (3rd edition), Newnes, Oxford, ISBN: 978-0323951975.

.

Why Learn about Control Registers?

The Mbed OS contains countless useful functions, which allow the programmer to write simple and effective code. This seems a good thing, but it is also sometimes limiting. What if we want to use a peripheral in a way not allowed by any of the functions, or to write code that is more efficient than the OS allows?

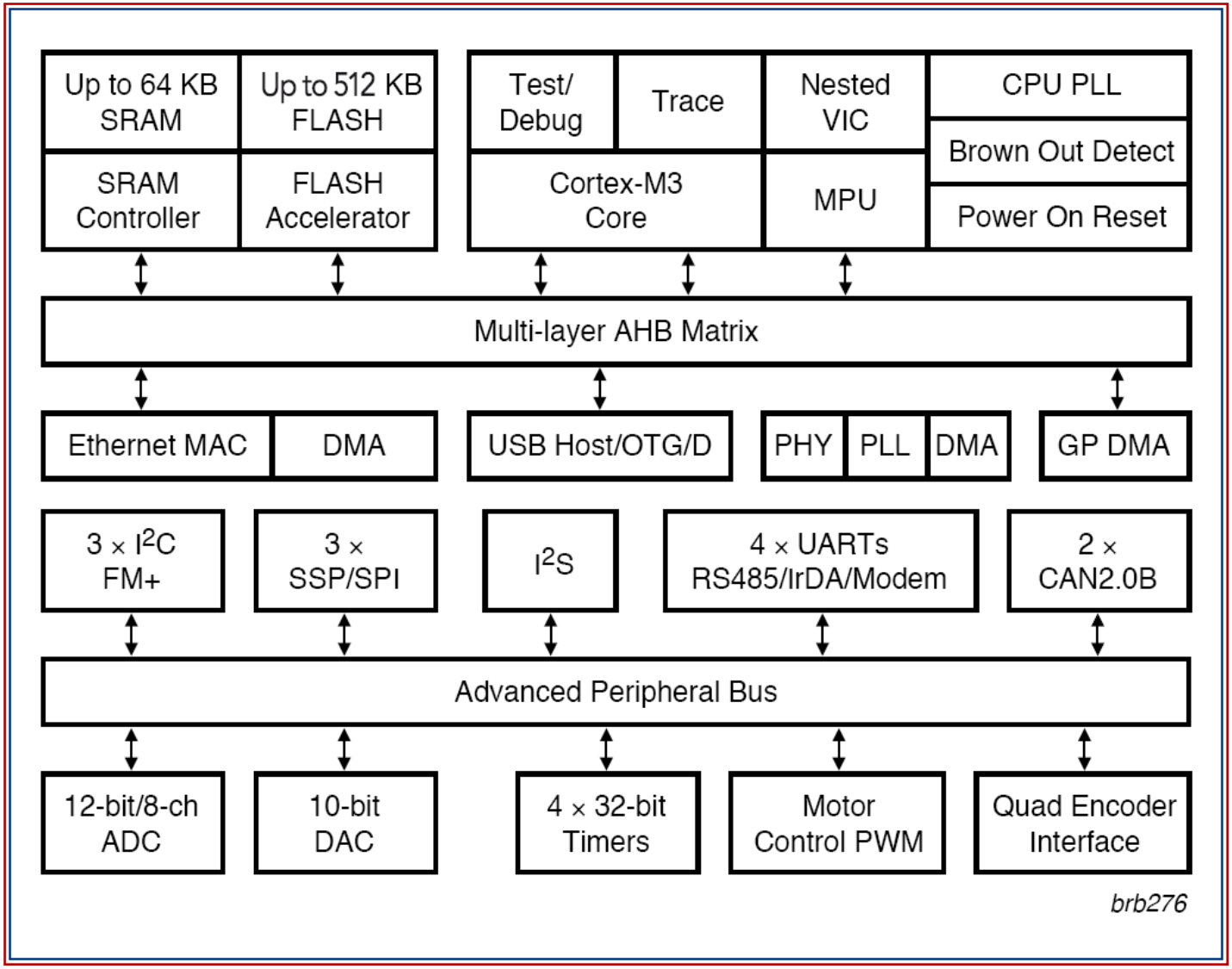
Therefore it is useful to understand how peripherals can be configured by direct access to the microcontroller's control registers. In turn, this leads to a deeper insight into some aspects of how a microcontroller works. As a further by-product, and because we will be working at the bit and byte level, this study develops further skills in C/C++ programming.

Note that this chapter deals only with the LPC1768.



Control Register Concepts

How does the CPU, running its program, communicate with all the peripherals on the microcontroller?



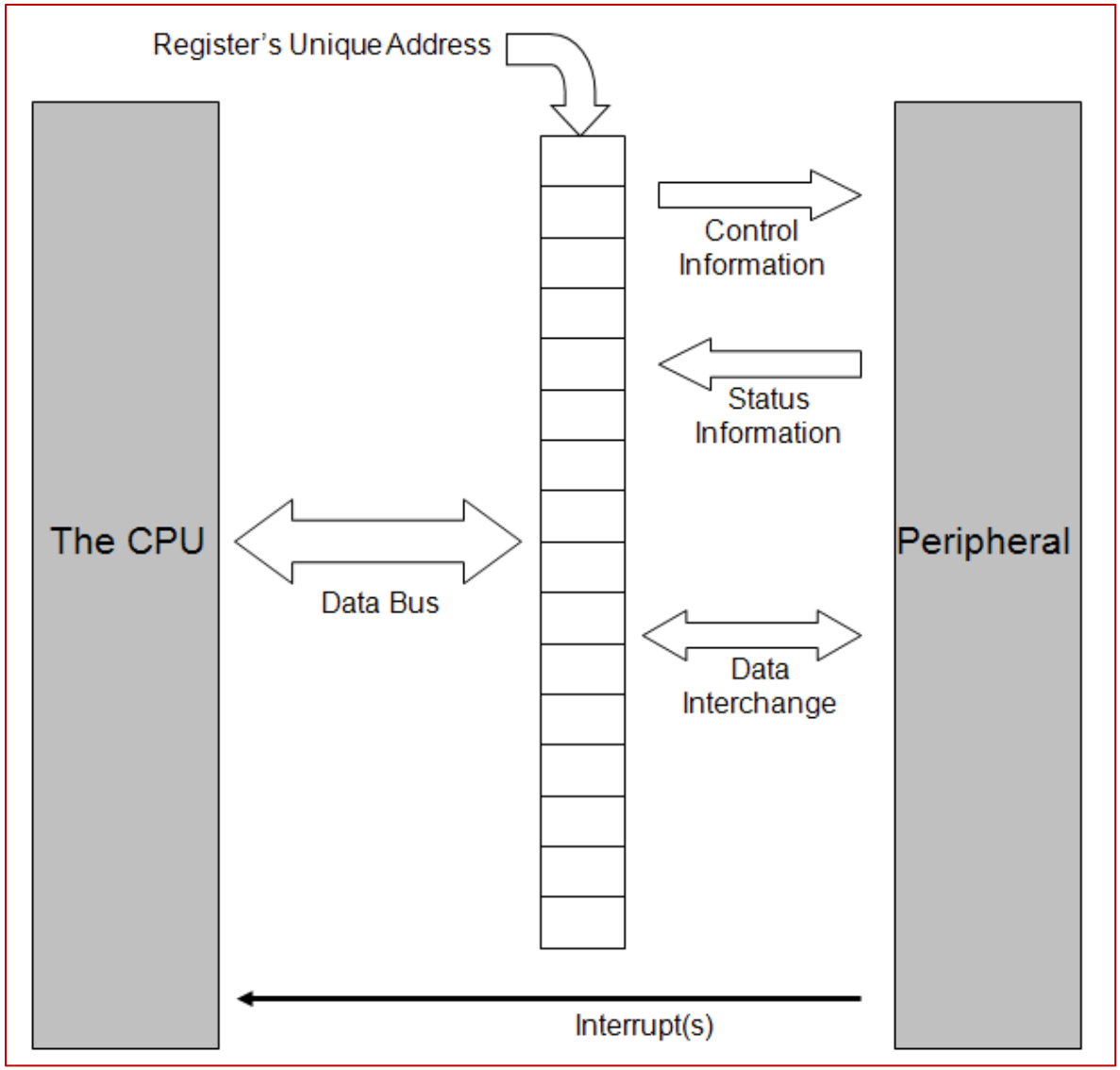
Control Register Concepts

Each microcontroller peripheral has one or more *system control registers* which act as the doorway between the peripheral and the CPU.

To the CPU, these registers look just like memory locations, and each has its own address in the memory map.

Each of the bits in the register is wired across to the peripheral. They might carry control or status information, or provide a path for data flow.

The microcontroller peripherals also usually generate interrupts, for example to flag when an ADC conversion is complete.



LPC1768 Digital I/O

The LPC1768 has its digital I/O arranged nominally as five 32-bit ports, that implies 160 bits in total. Only parts of these are used however, for example Port 0 has 28 bits implemented, Port 2 has 14, and Ports 3 and 4 have only two each.

In the end, among its 100 pins, the LPC1768 has around 70 general purpose I/O pins available. However the Mbed has only 40 accessible pins, so only a subset of the microcontroller pins actually appear on the Mbed interconnect, and many of these are shared with other features.

LPC1768 Digital I/O Control Registers

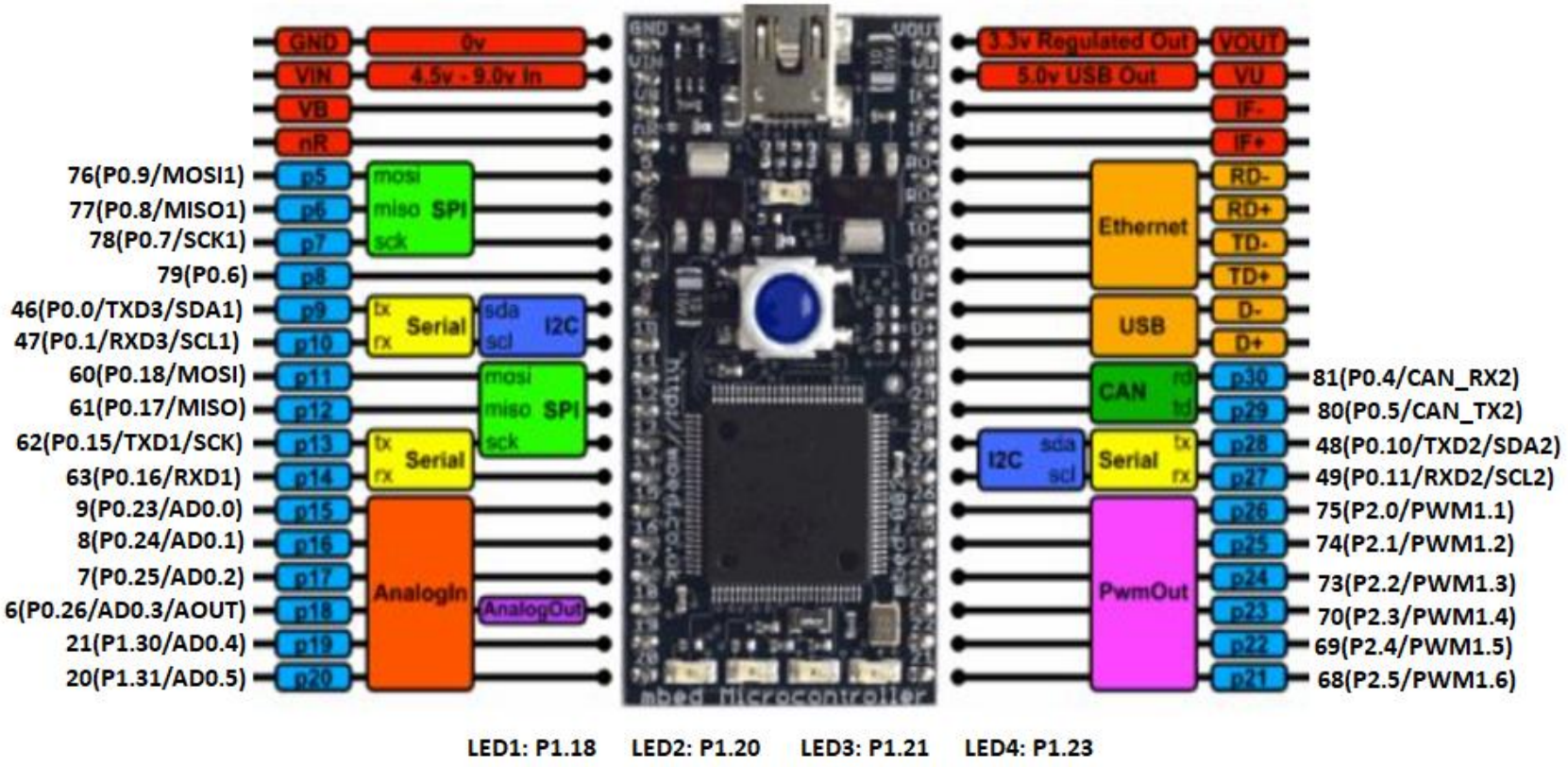
Using the control registers, it is possible to set each port pin as an input or as an output. Each port has a 32-bit register whose bits control the direction of each of its pins, called the **FIODIR** registers. There are also single-byte versions.

A second set of registers, called **FIOPIN**, hold the data value of the microcontroller's pins, whether they have been set as input or output.

Register Name	Register Function	Register Address
FIOnDIR	Sets the data direction of each pin in Port n, where n takes value 0 to 4. A port pin is set to output when its bit is set to 1, and to input when it is set to 0. Accessible as 32-bit word. Reset value = 0, i.e. all bits are set to input on reset.	-
FIO0DIR FIO1DIR FIO2DIR	Example of above for Port 0. Example of above for Port 1. Example of above for Port 2.	0x2009C000 0x2009C020 0x2009C040
FIOnDIRp	Sets the data direction of each pin in byte p of Port n, where p takes value 0 to 3. A port pin is set to output when its bit is set to 1. Accessible as byte.	-
FIO0DIR0 FIO0DIR1 FIO2DIR0	Example of above, Port 0 byte 0. Example of above, Port 0 byte 1. Example of above, Port 2 byte 0.	0x2009C000 0x2009C001 0x2009C040
FIO0PIN FIO1PIN FIO2PIN	Sets the data value of each bit in Port 0 or 2. Accessible as 32-bit word. Reset value = 0.	0x2009C014 0x2009C034 0x2009C054
FIO0PIN0 FIO0PIN1 FIO2PIN0	Sets the data value of each bit in selected byte of Port 0 or 2. Accessible as byte. Reset value = 0.	0x2009C014 0x2009C015 0x2009C054

Connecting LPC1768 Microcontroller pins to Mbed pins

Most of the Mbed pins connect directly to the LPC1768 microcontroller. These connections can be seen in the Mbed circuit diagram, and are summarised below. Against each Mbed pin which connects directly is shown the LPC1768 pin number, its port bit id, and any further function applied by the Mbed. For example pin 5 of the Mbed board connects to pin 76 of its microcontroller, and is bit 9 of Port 0.



A Digital Output Application

This program replicates the opening LED flashing program, without using the Mbed libraries. It switches port 2 bit 0 (and hence Mbed pin 26) high and low.

```
/*Program Example 14.1: Sets up a digital output pin using control registers, and
flashes an led.  For LPC1768 only.
                                                                              */
// function prototype
void delay(void);
//Define addresses of digital i/o control registers,
    //as pointers to volatile data
#define FIO2DIR0       (*(volatile unsigned char *)(0x2009C040))
#define FIO2PIN0       (*(volatile unsigned char *)(0x2009C054))

int main() {
  FIO2DIR0 = 0xFF;           // set port 2, lowest byte to output
  while(true) {
    FIO2PIN0 |= 0x01;        // Mbed pin 26. OR bit 0 with 1 to set pin high
    delay();
    FIO2PIN0 &= ~0x01;       // AND bit 0 with 0 to set pin low
    delay();
  }
}
//delay function
void delay(void){
  //loop variable j, volatile inhibits optimisation
  volatile int j;
  for (j = 0;j < 1000000;j++) {
    j++;
    j--;                      //waste time
  }
}
```

Digital Inputs

We can create digital inputs simply by setting a port bit to input, using the correct bit in an **FIODIR** register. This program example includes a digital input from a switch.

```
/* Program Example 14.3: Uses digital input and output using control registers, and
flashes an LED. LEDS connect to Mbed pins 25 and 26. Switch input to pin 7.
For LPC1768 only.
                                                */
//function prototypes
void delay(void);

//Define Digital I/O registers
#define FIO0DIR0 (*(volatile unsigned char *)(0x2009C000))
#define FIO0PIN0 (*(volatile unsigned char *)(0x2009C014))
#define FIO2DIR0 (*(volatile unsigned char *)(0x2009C040))
#define FIO2PIN0 (*(volatile unsigned char *)(0x2009C054))
//some variables
char a;
char b;
char i;

//continued on next slide
```

Digital Inputs (program example continued)

```
int main() {
  FIO0DIR0 = 0x00;              // set all bits of port 0 byte 0 to input
  FIO2DIR0 = 0xFF;              // set port 2 byte 0 to output
  while(true){
    if ((FIO0PIN0 & 0x08) == 0x08){   // bit test port 0 pin 7 (Mbed pin 7)
      a = 0x01;                 // this reverses the order of LED flashing
      b = 0x02;                 // based on the switch position
    }
    else {
      a = 0x02;
      b = 0x01;
    }
    FIO2PIN0 |= a;         //set port bit outputs based on value of a
    delay();
    FIO2PIN0 &= ~a;
    delay();
    for (i = 1;i <= 3;i++){
      FIO2PIN0 |= b;      //set port bit outputs based on value of b
      delay();
      FIO2PIN0 &= ~b;
      delay();
    }
  }                             //end while loop
}
//delay function
...
```

The Pin Select Register

The **PINSEL** register can allocate each pin to one of four possibilities. An example of part of one register, **PINSEL1**, is shown. This controls the upper half of Port 0.

Table 81. Pin function select register 1 (PINSEL1 - address 0x4002 C004) bit description

PINSEL1	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P0.16	GPIO Port 0.16	RXD1	SSEL0	SSEL	00
3:2	P0.17	GPIO Port 0.17	CTS1	MISO0	MISO	00
5:4	P0.18	GPIO Port 0.18	DCD1	MOSI0	MOSI	00
7:6	P0.19[1]	GPIO Port 0.19	DSR1	Reserved	SDA1	00
9:8	P0.20[1]	GPIO Port 0.20	DTR1	Reserved	SCL1	00
11:10	P0.21[1]	GPIO Port 0.21	RI1	Reserved	RD1	00
13:12	P0.22	GPIO Port 0.22	RTS1	Reserved	TD1	00
15:14	P0.23[1]	GPIO Port 0.23	AD0.0	I2SRX_CLK	CAP3.0	00
17:16	P0.24[1]	GPIO Port 0.24	AD0.1	I2SRX_WS	CAP3.1	00
19:18	P0.25	GPIO Port 0.25	AD0.2	I2SRX_SDA	TXD3	00
21:20	P0.26	GPIO Port 0.26	AD0.3	AOUT	RXD3	00
23:22	P0.27[1][2]	GPIO Port 0.27	SDA0	USB_SDA	Reserved	00
25:24	P0.28[1][2]	GPIO Port 0.28	SCL0	USB_SCL	Reserved	00
27:26	P0.29	GPIO Port 0.29	USB_D+	Reserved	Reserved	00
29:28	P0.30	GPIO Port 0.30	USB_D–	Reserved	Reserved	00
31:30	-	Reserved	Reserved	Reserved	Reserved	00

(From LPC17xxx User Manual Rev. 3.1, UM10360)

The Pin Mode Register

As an example pin mode select register, **PINMODE0** is shown. This controls the input characteristics of the lower half of Port 0. The pattern is the same for every pin, so there's no need for repetition.
Pull-up and pull-down resistors can be enabled. The repeater mode is a facility which enables a pull-up resistor when the input is a Logic 1, and a pull-down if it's low. If the external circuit changes so that the input is no longer driven, then the input will hold its most recent value.

Table 88. Pin Mode select register 0 (PINMODE0 - address 0x4002 C040) bit description

PINMODE0	Symbol	Value	Description	Reset value
1:0	P0.00MODE		Port 0 pin 0 on-chip pull-up/down resistor control.	00
		00	P0.0 pin has a pull-up resistor enabled.	
		01	P0.0 pin has repeater mode enabled.	
		10	P0.0 pin has neither pull-up nor pull-down.	
		11	P0.0 has a pull-down resistor enabled.	
3:2	P0.01MODE		Port 0 pin 1 control, see P0.00MODE.	00

Power Control Registers

To conserve power, it is possible to turn off power and clock to many of the LPC1768 peripherals. This power management is controlled by the **PCONP** register, seen in part here. Where a bit is set to 1, the peripheral is enabled.

Table 46. Power Control for Peripherals register (PCONP - address 0x400F C0C4) bit description

Bit	Symbol	Description	Reset value
0	-	Reserved.	NA
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	-	Reserved.	NA
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I^2C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1
10	PCSSP1	The SSP 1 interface power/clock control bit.	1
11	-	Reserved.	NA
12	PCADC	A/D converter (ADC) power/clock control bit. **Note:** Clear the PDN bit in the AD0CR before clearing this bit, and set this bit before setting PDN.	0

Clock Select Registers

Some control is possible over the peripherals' clock frequency, using the **PCLKSEL** registers. This will control a peripheral's speed of operation and hence its power consumption. Peripheral clocks are taken from the clock which drives the CPU, called **CCLK**. For the mbed, **CCLK** runs at 96 MHz. Partial details of **PCLKSEL0** are shown. Two bits are used per peripheral to control the clock frequency to each. The four possible combinations are also shown.

Table 40. Peripheral Clock Selection register 0 (PCLKSEL0 - address 0x400F C1A8) bit description

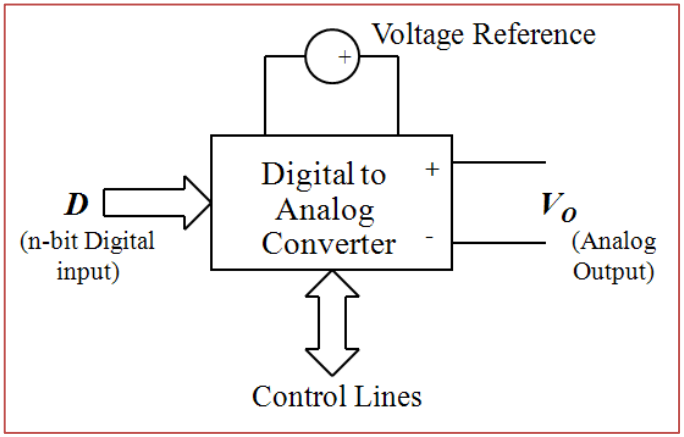
Bit	Symbol	Description	Reset value
1:0	PCLK_WDT	Peripheral clock selection for WDT.	00
3:2	PCLK_TIMER0	Peripheral clock selection for TIMER0.	00
5:4	PCLK_TIMER1	Peripheral clock selection for TIMER1.	00
7:6	PCLK_UART0	Peripheral clock selection for UART0.	00
9:8	PCLK_UART1	Peripheral clock selection for UART1.	00
11:10	-	Reserved.	NA
13:12	PCLK_PWM1		
15:14	PCLK_I2C0		
17:16	PCLK_SPI		
19:18	-		
21:20	PCLK_SSP1		
23:22	PCLK_DAC		
25:24	PCLK_ADC		
27:26	PCLK_CAN1		
29:28	PCLK_CAN2		
31:30	PCLK_ACF		

Table 42. Peripheral Clock Selection register bit values

PCLKSEL0 and PCLKSEL1 individual peripheral's clock select options	Function	Reset value
00	PCLK_peripheral = CCLK/4	00
01	PCLK_peripheral = CCLK	
10	PCLK_peripheral = CCLK/2	
11	PCLK_peripheral = CCLK/8, except for CAN1, CAN2, and CAN filtering when "11" selects = CCLK/6.	

[1] PCLK_CAN1 and PCLK_CAN2 must have the same PCLK divide value when the CAN function is used.

Using the DAC



The LPC1768 DAC has a set of control registers. DAC power is always enabled, so there is no need to consider the **PCONP** register.

The *only* pin that the DAC output is available on is Port 0 pin 26, so this must be allocated through **PINSEL1** (where DAC output is labelled AOUT). This pin is connected to mbed pin 18.

The only register specific to the DAC that we use is the **DACR** register, shown here. On the LPC1768 the reference voltage is applied between terminals labelled V_{REFP} and V_{REFN}; these are connected to 3.3 V and 0 V respectively.

Table 540: D/A Converter Register (DACR - address 0x4008 C000) bit description

Bit	Symbol	Value	Description	Reset Value
5:0	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:6	VALUE		After the selected settling time after this field is written with a new VALUE, the voltage on the AOUT pin (with respect to V_{SSA}) is VALUE × ((V_{REFP} - V_{REFN})/1024) + V_{REFN}.	0
16	BIAS[1]	0	The settling time of the DAC is 1 μs max, and the maximum current is 700 μA. This allows a maximum update rate of 1 MHz.	0
		1	The settling time of the DAC is 2.5 μs and the maximum current is 350 μA. This allows a maximum update rate of 400 kHz.	
31:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

[1] The settling times noted in the description of the BIAS bit are valid for a capacitance load on the AOUT pin not exceeding 100 pF. A load impedance value greater than that value will cause settling time longer than the specified time. One or more graph(s) of load impedance vs. settling time will be included in the final data sheet.

A DAC Application

This program example replicates the simple sawtooth waveform, first achieved in Chapter 4. An integer variable, **dac_value**, is repeatedly incremented and transferred to the DAC input, in register **DACR**. It has to be shifted left 6 times, to place it in the correct bits of the **DACR** register.

```
/* Program Example 14.4: Sawtooth waveform on DAC output. View on oscilloscope.
Port 0.26 is used for DAC output, i.e. Mbed Pin 18
For LPC1768 only.                                                    */

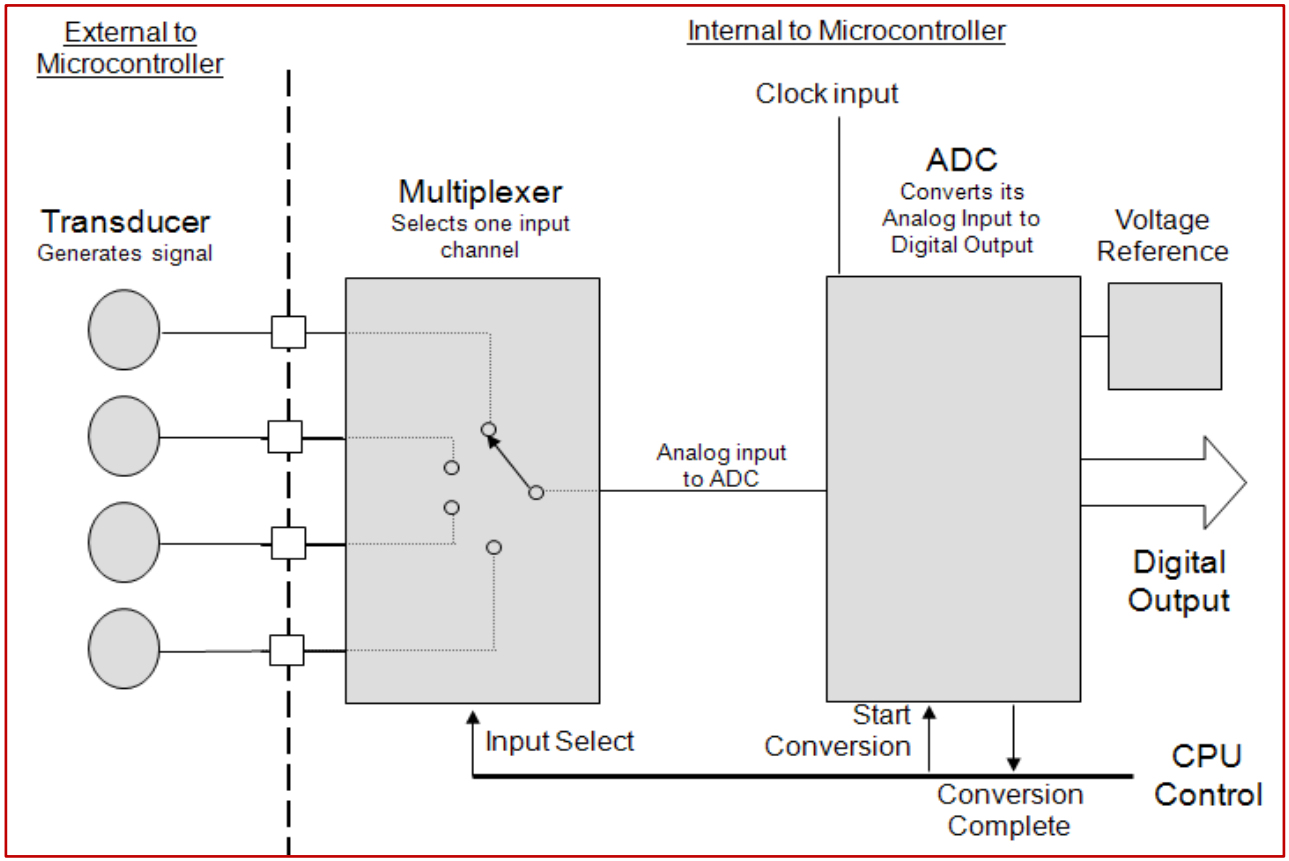
// function prototype
void delay(void);
// variable declarations
int dac_value;              //the value to be output
//define addresses of control registers, as pointers to volatile data
#define DACR (*(volatile unsigned long *)(0x4008C000))
#define PINSEL1 (*(volatile unsigned long *)(0x4002C004))

int main(){
  PINSEL1 = 0x00200000; //set bits 21-20 to 10 to enable analog out on P0.26
  //or use: PINSEL1 = PINSEL1 | (1<<21); //(all bits in register reset to 0)
  while(true){
    for (dac_value = 0;dac_value < 1024;dac_value = dac_value + 4){
      DACR = (dac_value << 6);
      delay();
    }
  }
}

//delay function
...
```

Using the ADC

Controlling the ADC through its registers includes selecting or applying the voltage reference, clock speed, input channel, starting a conversion, detecting a completion, and reading the output data. The LPC1768 has a number of registers which control its ADC. Only two are applied here, the ADC control register, **ADCR,** and the Global Data Register, **ADGDR**. As the ADC is switched off on power-up, it needs to be enabled through bit 12 in the **PCONP** register.



The ADC Control Register

Table 532: A/D Control Register (AD0CR - address 0x4003 4000) bit description

Bit	Symbol	Value	Description	Reset value
7:0	SEL		Selects which of the AD0.7:0 pins is (are) to be sampled and converted. For AD0, bit 0 selects Pin AD0.0, and bit 7 selects pin AD0.7. In software-controlled mode, only one of these bits should be 1. In hardware scan mode, any value containing 1 to 8 ones is allowed. All zeroes is equivalent to 0x01.	0x01
15:8	CLKDIV		The APB clock (PCLK_ADC0) is divided by (this value plus one) to produce the clock for the A/D converter, which should be less than or equal to 13 MHz. Typically, software should program the smallest value in this field that yields a clock of 13 MHz or slightly less, but in certain cases (such as a high-impedance analog source) a slower clock may be desirable.	0
16	BURST	1	The AD converter does repeated conversions at up to 200 kHz, scanning (if necessary) through the pins selected by bits set to ones in the SEL field. The first conversion after the start corresponds to the least-significant 1 in the SEL field, then higher numbered 1-bits (pins) if applicable. Repeated conversions can be terminated by clearing this bit, but the conversion that's in progress when this bit is cleared will be completed. **Remark:** START bits must be 000 when BURST = 1 or conversions will not start.	0
		0	Conversions are software controlled and require 65 clocks.	
20:17	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
21	PDN	1	The A/D converter is operational.	0
		0	The A/D converter is in power-down mode.	
23:22	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
26:24	START		When the BURST bit is 0, these bits control whether and when an A/D conversion is started:	0
		000	No start (this value should be used when clearing PDN to 0).	
		001	Start conversion now.	

Note: further more advanced options for START control are available, see full Table

The ADC Global Data Register

Table 533: A/D Global Data Register (AD0GDR - address 0x4003 4004) bit description

Bit	Symbol	Description	Reset value
3:0	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
15:4	RESULT	When DONE is 1, this field contains a binary fraction representing the voltage on the AD0[n] pin selected by the SEL field, as it falls within the range of V_{REFP} to V_{REFN}. Zero in the field indicates that the voltage on the input pin was less than, equal to, or close to that on V_{REFN}, while 0xFFF indicates that the voltage on the input was close to, equal to, or greater than that on V_{REFP}.	NA
23:16	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
26:24	CHN	These bits contain the channel from which the RESULT bits were converted (e.g. 000 identifies channel 0, 001 channel 1...).	NA
29:27	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
30	OVERRUN	This bit is 1 in burst mode if the results of one or more conversions was (were) lost and overwritten before the conversion that produced the result in the RESULT bits. This bit is cleared by reading this register.	0
31	DONE	This bit is set to 1 when an A/D conversion completes. It is cleared when this register is read and when the ADCR is written. If the ADCR is written while a conversion is still in progress, this bit is set and a new conversion is started.	0

An ADC Bargraph Application (initialisation)

```
/* Program Example 14.5: A bar graph meter for ADC input, using control registers to
set up ADC and digital I/O
                                                                    */
// variable declarations
char ADC_channel=1;     // ADC channel 1
int  ADCdata;           //this will hold the result of the conversion
int DigOutData=0;       //a buffer for the output display pattern

// function prototype
void delay(void);

//define addresses of control registers, as pointers to volatile data
//(i.e. the memory contents)
#define PINSEL1         (*(volatile unsigned long *)(0x4002C004))
#define PCONP           (*(volatile unsigned long *)(0x400FC0C4))
#define AD0CR           (*(volatile unsigned long *)(0x40034000))
#define AD0GDR          (*(volatile unsigned long *)(0x40034004))
#define FIO2DIR0        (*(volatile unsigned char *)( 0x2009C040))
#define FIO2PIN0        (*(volatile unsigned char *)( 0x2009C054))

int main() {
  FIO2DIR0=0xFF;// set lower byte of Port 2 to output, this drives bar graph

//initialise the ADC
  PINSEL1=0x00010000; //set bits 17-16 to 01 to enable AD0.1 (mbed pin 16)
  PCONP |= (1 << 12);                 // enable ADC clock
  AD0CR =   (1 << ADC_channel)        // select channel 1
          | (4 << 8)        // Divide incoming clock by (4+1), giving 4.8MHz
          | (0 << 16)       // BURST = 0, conversions under software control
          | (1 << 21)       // PDN = 1, enables power
          | (1 << 24);      // START = 1, start A/D conversion now
 ...
```

An ADC Bargraph Application (main while(true) loop)

```
while(true) {                        // infinite loop
    AD0CR = AD0CR | 0x01000000;   //start conversion by setting bit 24 to 1,
                                                          //by ORing
    // wait for it to finish by polling the ADC DONE bit
    while ((AD0GDR & 0x80000000) == 0) {  //test DONE bit, wait till it's 1
    }
    ADCdata = AD0GDR;             // get the data from AD0GDR
    AD0CR &= 0xF8FFFFFF;          //stop ADC by setting START bits to zero
   // Shift data 4 bits to right justify, and 2 more to give 10-bit ADC
   // value - this gives convenient range of just over one thousand.
    ADCdata=(ADCdata>>6)&0x03FF;    //and mask
    DigOutData=0x00;                //clear the output buffer
    //display the data
    if (ADCdata>200)
      DigOutData=(DigOutData|0x01);  //set the lsb by ORing with 1
    if (ADCdata>400)
      DigOutData=(DigOutData|0x02);  //set the next lsb by ORing with 1
    if (ADCdata>600)
      DigOutData=(DigOutData|0x04);
    if (ADCdata>800)
      DigOutData=(DigOutData|0x08);
    if (ADCdata>1000)
      DigOutData=(DigOutData|0x10);

    FIO2PIN0 = DigOutData;        // set port 2 to Digoutdata
    delay();      // pause
   }
}

    void delay(void){                   //delay function.
//program continues
```

Changing ADC Conversion Speed

One of the limitations of the original Mbed **AnalogIn** API is the slow speed of conversion. This can be varied by adjusting the ADC clock speed. Program Example 14.6 replicates Program Example 5.5, and combines ADC, DAC and digital I/O, illustrating how these can be used together.

```
/* Program Example 14.6: Explore ADC conversion times, programming control registers
directly. ADC value is transferred to DAC, while an output pin is strobed to indicate
conversion duration. Observe on oscilloscope
                                                                              */
….
….
  while(true){          // infinite loop
    // start A/D conversion by modifying bits in the AD0CR register
    AD0CR &= (AD0CR & 0xFFFFFF00);
    FIO2PIN0 |= 0x01;                      // OR bit 0 with 1 to set pin high
    AD0CR |= (1 << ADC_channel) | (1 << 24);
    // wait for it to finish by polling the ADC DONE bit
    while((AD0GDR & 0x80000000) == 0) {
    }
    FIO2PIN0 &= ~0x01;                    // AND bit 0 with 0 to set pin low

    ADCdata = AD0GDR;               // get the data from AD0GDR
    AD0CR &= 0xF8FFFFFF;            //stop ADC by setting START bits to zero

   // shift data 4 bits to right justify, and 2 more to give 10-bit ADC value
    ADCdata=(ADCdata>>6)&0x03FF;    //and mask
    DACR=(ADCdata<<6);      //could be merged with previous line,
                               // but separated for clarity
    //delay();                //insert delay if wished
  }
}
…
```

A Conclusion on Using the Control Registers

- We have explored the use of the LPC1768 control registers, in connection with use of the digital I/O, ADC and DAC peripherals.
- We have demonstrated how these registers allow the peripherals to be controlled directly, without using the mbed libraries. This has allowed greater flexibility of use of the peripherals, at the cost of getting into the tiny detail of the registers, and programming at the level of the bits that make them up.
- Ordinarily we probably wouldn't want to program like this; it's time-consuming, inconvenient and error-prone. However if we need a configuration or setting not offered by the mbed libraries, this approach can be a way forward.
- While we've only worked in this way in connection with three of the peripherals, it's possible to do it with any of them. The three used are some of the simpler; others require even more attention to detail.

Chapter Review

- In this chapter we have recognised a different way of controlling the mbed peripherals. It demands a much deeper understanding of the mbed microcontroller, but allows for much greater flexibility.
- There are registers which relate just to one peripheral, and others which relate to microcontroller performance as a whole.
- We have begun to implement features that are not currently available in the mbed library, for example in the change of the ADC conversion speed.
- The chapter only introduces a small range of the control registers which are used by the LPC1768. However it should have given you the confidence to look up and begin to apply any that you need.

