Lesson ADC + PWM

- Pin Selection and Pin Mode
- ADC (Analog to Digital Converter)
- PWM (Pulse Width Modulation)
- Lab: ADC + PWM

Pin Selection and Pin Mode

Objective

Know how to select a specific functionality of a given LPC40xx pin. Know how to select a pin mode.

Pin Selection

Every GPIO pin of the LPC40xx is capable of other alternative functionalities. Pin selection is the method by which a user is able to designate the functionality of any given pin. For example, GPIO Pin 0.0 can alternatively be used for CAN channel 1 receive, UART channel 3 transmit, and I2C channel 1 data line.

Table 84. Type D I/O Control registers: FUNC values and pin functions

	Value of FUNC field in IOCON register							
Register	000	001	010	011	100	101	110	111
IOCON_P0_0	P0[0]	CAN_RD1	U3_TXD	I2C1_SDA	U0_TXD			
IOCON_P0_1	P0[1]	CAN_TD1	U3_RXD	I2C1_SCL	U0_RXD			
IOCON_P0_2	P0[2]	U0_TXD	U3_TXD					
IOCON_P0_3	P0[3]	U0_RXD	U3_RXD					
IOCON_P0_4	P0[4]	I2S_RX_SCK	CAN_RD2	T2_CAP0		CMP_ROSC		LCD_VD[0]
IOCON_P0_5	P0[5]	I2S_RX_WS	CAN_TD2	T2_CAP1		CMP_RESET		LCD_VD[1]
IOCON_P0_6	P0[6]	I2S_RX_SDA	SSP1_SSEL	T2_MAT0	U1_RTS	CMP_ROSC		LCD_VD[8]
IOCON_P0_10	P0[10]	U2_TXD	I2C2_SDA	T3_MAT0				LCD_VD[5]
IOCON_P0_11	P0[11]	U2_RXD	I2C2_SCL	T3_MAT1				LCD_VD[10]
IOCON_P0_14	P0[14]	USB_HSTEN2	SSP1_SSEL	USB_CONNECT2				
IOCON_P0_15	P0[15]	U1_TXD	SSP0_SCK			SPIFI_IO[2]		
IOCON_P0_16	P0[16]	U1_RXD	SSP0_SSEL			SPIFI_IO[3]		
IOCON_P0_17	P0[17]	U1_CTS	SSP0_MISO			SPIFI_IO[1]		
IOCON_P0_18	P0[18]	U1_DCD	SSP0_MOSI			SPIFI_IO[0]		
IOCON_P0_19	P0[19]	U1_DSR	SD_CLK	I2C1_SDA				LCD_VD[13]
IOCON_P0_20	P0[20]	U1_DTR	SD_CMD	I2C1_SCL				LCD_VD[14]
IOCON_P0_21	P0[21]	U1_RI	SD_PWR	U4_OE	CAN_RD1	U4_SCLK		
IOCON_P0_22	P0[22]	U1_RTS	SD_DAT[0]	U4_TXD	CAN_TD1	SPIFI_CLK		
IOCON_P1_0	P1[0]	ENET_TXD0		T3_CAP1	SSP2_SCK			
IOCON_P1_1	P1[1]	ENET_TXD1		T3_MAT3	SSP2_MOSI			
IOCON_P1_2	P1[2]	ENET_TXD2	SD_CLK	PWM0[1]				
IOCON_P1_3	P1[3]	ENET_TXD3	SD_CMD	PWM0[2]				
IOCON_P1_4	P1[4]	ENET_TX_EN		T3_MAT2	SSP2_MISO			
IOCON_P1_8	P1[8]	ENET_CRS		T3_MAT1	SSP2_SSEL			
IOCON_P1_9	P1[9]	ENET_RXD0		T3_MAT0				
IOCON_P1_10	P1[10]	ENET_RXD1		T3_CAP0				
IOCON_P1_11	P1[11]	ENET_RXD2	SD_DAT[2]	PWM0[6]				
IOCON_P1_12	P1[12]	ENET_RXD3	SD_DAT[3]	PWM0_CAP0		CMP1_OUT		
IOCON_P1_13	P1[13]	ENET_RX_DV						

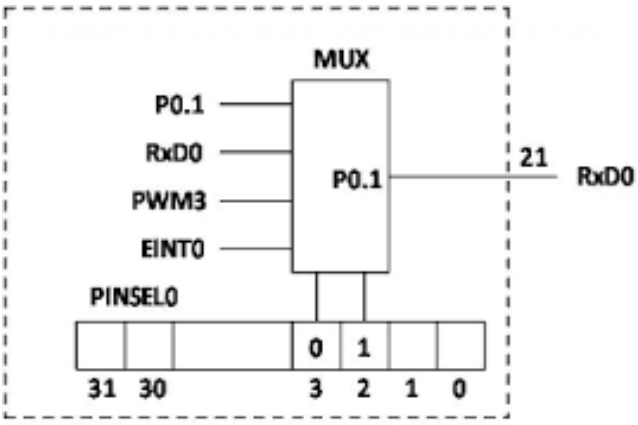


Figure 1B. *I/O Pin Select Mux (from LPC2148, for illustration purposes only)*

In order to select the I2C2_SDA functionality of pin 0.10, one must set bit 1, reset bit 0 & 3 of the IOCON register function field to 010.

```
// Using LPC40xx.h pointers
LPC_IOCON->P0_10 &= ~0b010;// reset all bits of function[2:0]
LPC_IOCON->P0_10 |= 0b010;// set the function bit for I2C2
```

Pin Mode

The LPC17xx has several registers dedicated to setting a pin's mode. Mode refers to enabling/disabling pull up/down resistors as well as open-drain configuration. PINMODE registers allow users to enable a pull-up (00), enable pull-up and pull-down (01), disable pull-up and pull-down (10), and enable pull-down (11). PINMC

PINMODE0	Pin mode select register 0	R/W	0	0x4002 C040
PINMODE1	Pin mode select register 1	R/W	0	0x4002 C044
PINMODE2	Pin mode select register 2	R/W	0	0x4002 C048
PINMODE3	Pin mode select register 3.	R/W	0	0x4002 C04C
PINMODE4	Pin mode select register 4	R/W	0	0x4002 C050
PINMODE5	Pin mode select register 5	R/W	0	0x4002 C054
PINMODE6	Pin mode select register 6	R/W	0	0x4002 C058
PINMODE7	Pin mode select register 7	R/W	0	0x4002 C05C
PINMODE9	Pin mode select register 9	R/W	0	0x4002 C064
PINMODE_OD0	Open drain mode control register 0	R/W	0	0x4002 C068
PINMODE_OD1	Open drain mode control register 1	R/W	0	0x4002 C06C
PINMODE_OD2	Open drain mode control register 2	R/W	0	0x4002 C070
PINMODE_OD3	Open drain mode control register 3	R/W	0	0x4002 C074
PINMODE_OD4	Open drain mode control register 4	R/W	0	0x4002 C078

Figure 2. LPC17xx User Manual PINMODE & PINMODE_OD

Table 87. 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
5:4	P0.02MODE		Port 0 pin 2 control, see P0.00MODE.	00
7:6	P0.03MODE		Port 0 pin 3 control, see P0.00MODE.	00
9:8	P0.04MODE[1]		Port 0 pin 4 control, see P0.00MODE.	00
11:10	P0.05MODE[1]		Port 0 pin 5 control, see P0.00MODE.	00
13:12	P0.06MODE		Port 0 pin 6 control, see P0.00MODE.	00
15:14	P0.07MODE		Port 0 pin 7 control, see P0.00MODE.	00
17:16	P0.08MODE		Port 0 pin 8 control, see P0.00MODE.	00
19:18	P0.09MODE		Port 0 pin 9control, see P0.00MODE.	00
21:20	P0.10MODE		Port 0 pin 10 control, see P0.00MODE.	00
23:22	P0.11MODE		Port 0 pin 11 control, see P0.00MODE.	00
29:24	-		Reserved.	NA
31:30	P0.15MODE		Port 0 pin 15 control, see P0.00MODE.	00

Table 94. Open Drain Pin Mode select register 0 (PINMODE_OD0 - address 0x4002 C068) bit description

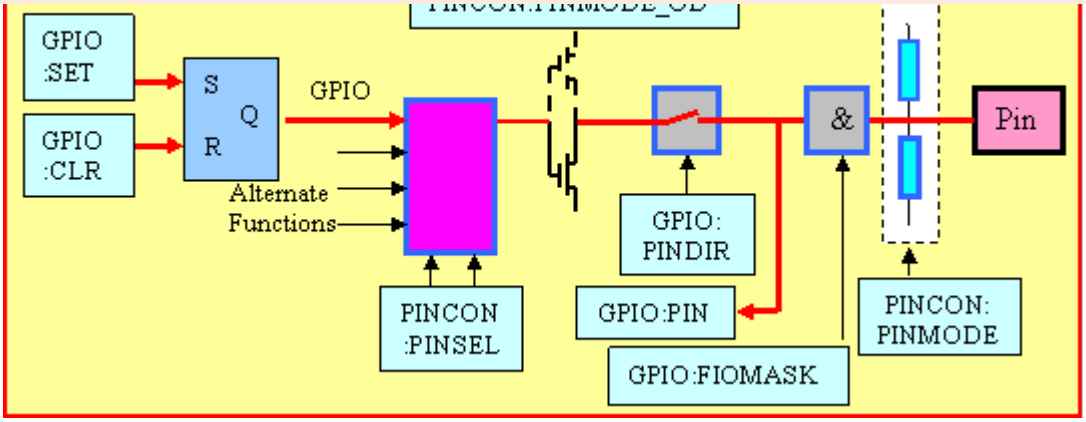
PINMODE_OD0	Symbol	Value	Description	Reset value
0	P0.00OD[3]		Port 0 pin 0 open drain mode control.	0
		0	P0.0 pin is in the normal (not open drain) mode.	
		1	P0.0 pin is in the open drain mode.	
1	P0.01OD[3]		Port 0 pin 1 open drain mode control, see P0.00OD	0
2	P0.02OD		Port 0 pin 2 open drain mode control, see P0.00OD	0
3	P0.03OD		Port 0 pin 3 open drain mode control, see P0.00OD	0
4	P0.04OD		Port 0 pin 4 open drain mode control, see P0.00OD	0
5	P0.05OD		Port 0 pin 5 open drain mode control, see P0.00OD	0
6	P0.06OD		Port 0 pin 6 open drain mode control, see P0.00OD	0
7	P0.07OD		Port 0 pin 7 open drain mode control, see P0.00OD	0
8	P0.08OD		Port 0 pin 8 open drain mode control, see P0.00OD	0
9	P0.09OD		Port 0 pin 9 open drain mode control, see P0.00OD	0

Figure 4. LPC17xx User Manual PINMODE_OD0

For example, if one desires to configure pin 0.09 to enable a pull-up resistor and open drain mode, one must clear bits 18 & 19 of PINMODE0 register, and set bit 9 of register PINMODE_OD0.

```
// Using the memory address from the datasheet
*(0x4002C040) &= ~(0x3 << 18);□□// Clear bits 18 & 19
*(0x4002C068) |= (0x1 << 9);□□// Set bit 9
// Using LPC17xx.h pointers
LPC_PINCON->PINMODE0 &= ~(0x3 << 18);□□// Clear bits 18 & 19
LPC_PINCON->PINMODE_OD0 |= (0x1 << 9);□□// Set bit 9
```

> You may find it helpful to automate register setting and/or clearing. Per our Coding Standards, inline functions should be used (not Macros).



?

Figure 5. LPC17xx Pin Registers & Circuit (credit: https://sites.google.com/site/johnkneenmicrocontrollers/input_output/io_1768)

ADC (Analog to Digital Converter)

Objective

To learn about the use of ADCs, their different types, their related parameters, and how to set up an ADC driver for the LPC40xx.

What does ADC accomplish?

An Analog to Digital Converter is needed whenever one needs to interface a digital system with an analog device. For example, if one needs to read the voltage across a resistor, and use the value within an algorithm running on the SJOne board, an ADC circuit is needed to convert the analog voltage to a discrete digital value. Luckily, the LPC40xx, like most microcontrollers, includes an ADC circuit that we can utilize.

Different types of ADC circuits

Flash ADC

The simplest and fastest ADC circuit relies on a series of comparators that compare the input voltage to a range of voltage reference values. The digital output of the comparators is wired to a priority encoder. The output of the priority encoder represents the binary value of the input voltage.

Note that the number of bits of the binary output (n) requires (2^n-1) comparators. Therefore, the circuit complexity grows exponentially with respect to the number of bits used to represent the converted value (resolution).

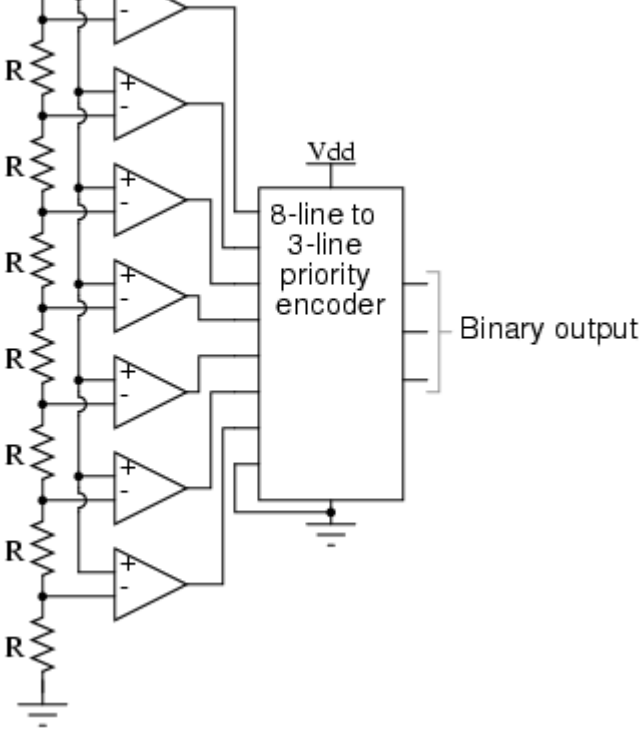


Figure 1. Flash ADC Circuit (credit: allaboutcircuits.com)

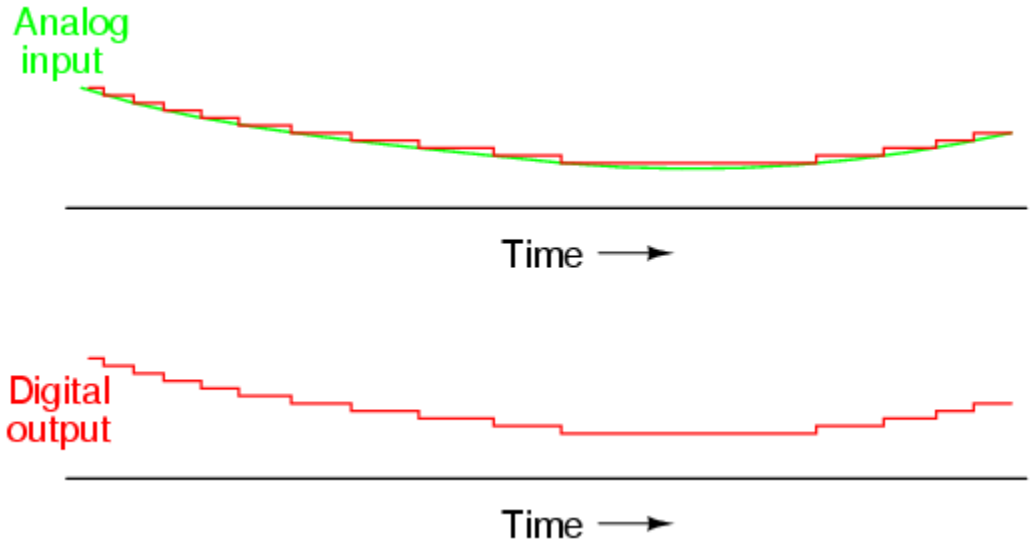


Figure 2. Flash ADC Timing (credit: allaboutcircuits.com)

Digital Ramp ADC

This type of ADC utilizes an up counter, a comparator, a DAC, and a register. DACs (Digital Analog Converters), as their name suggests, perform the inverse operation of an ADC, i.e. They convert a binary input into an analog voltage output. The up counter starts at zero and counts up synchronously. The output of the counter is wired to the DAC. The analog output of the DAC is compared to the analog input signal. As long as the comparator indicates that the input voltage is larger than the DAC's value, the counter continues to increment. Eventually, the DAC's output will exceed the input voltage, and the comparator will activate the counter's reset signal as well as the register's load signal. The register's output represents the binary value of the input analog signal.

Note that be to produce the

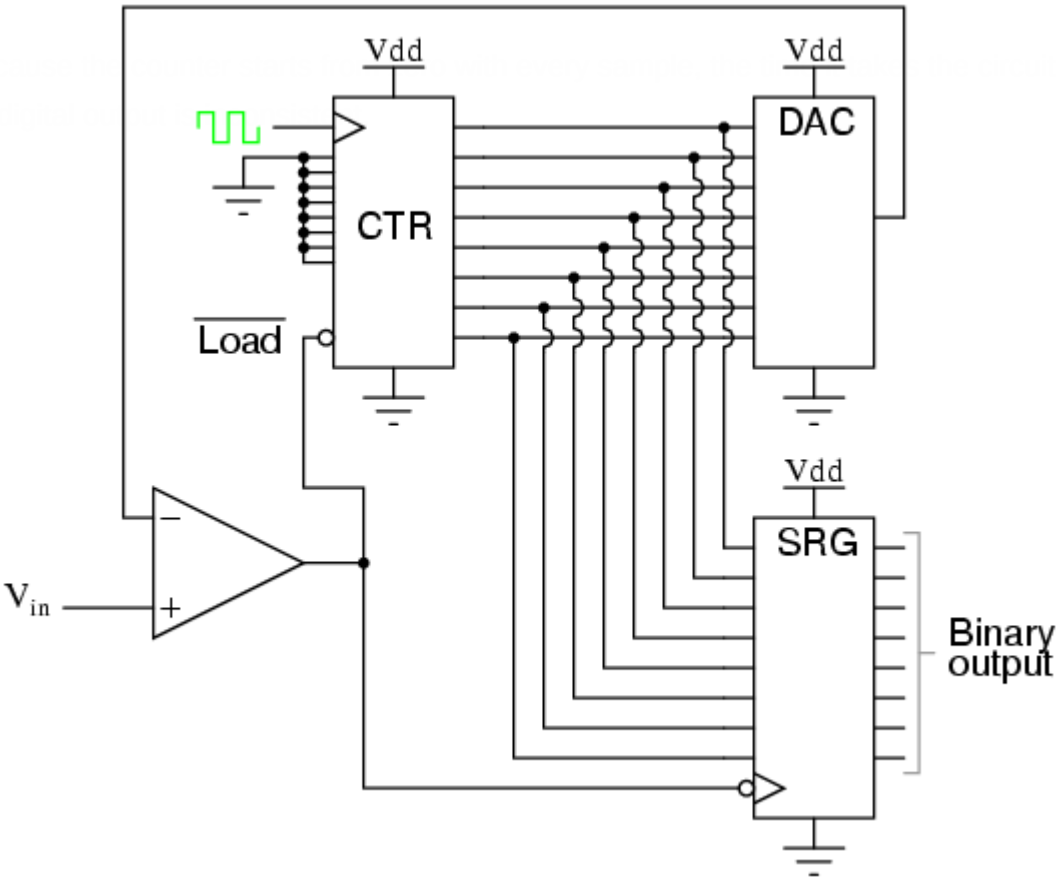


Figure 3. Digital Ramp ADC Circuit (credit: allaboutcircuits.com)

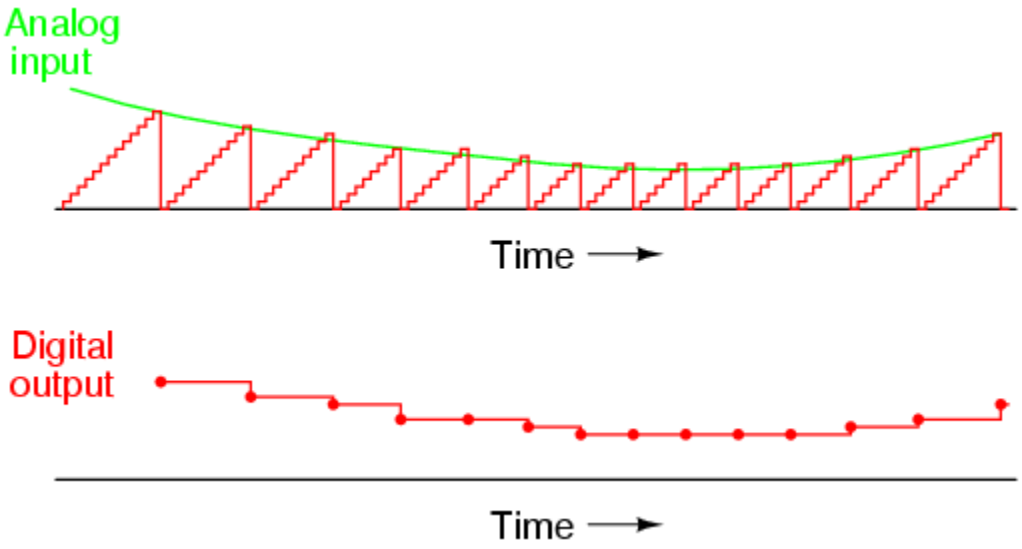


Figure 4a. Digital Ramp ADC Timing (credit: allaboutcircuits.com)

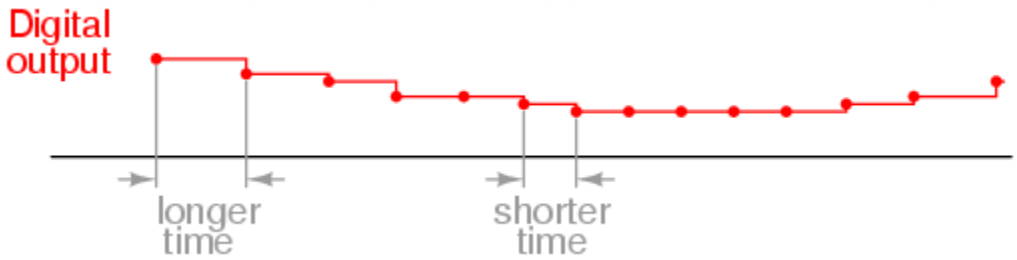


Figure 4b. Digital Ramp ADC Timing Variance (credit: allaboutcircuits.com)

Successive Approximation ADC

A successive approximation ADC works very similarly to a digital ramp ADC, except it utilizes a successive approximation register (SAR) in place of the counter. The SAR sets each bit from MSB to LSB according to its greater/less than logic input signal.

This type of ADC is more popular than flash and digital ramp due to its consistent timing and relatively scalable design.

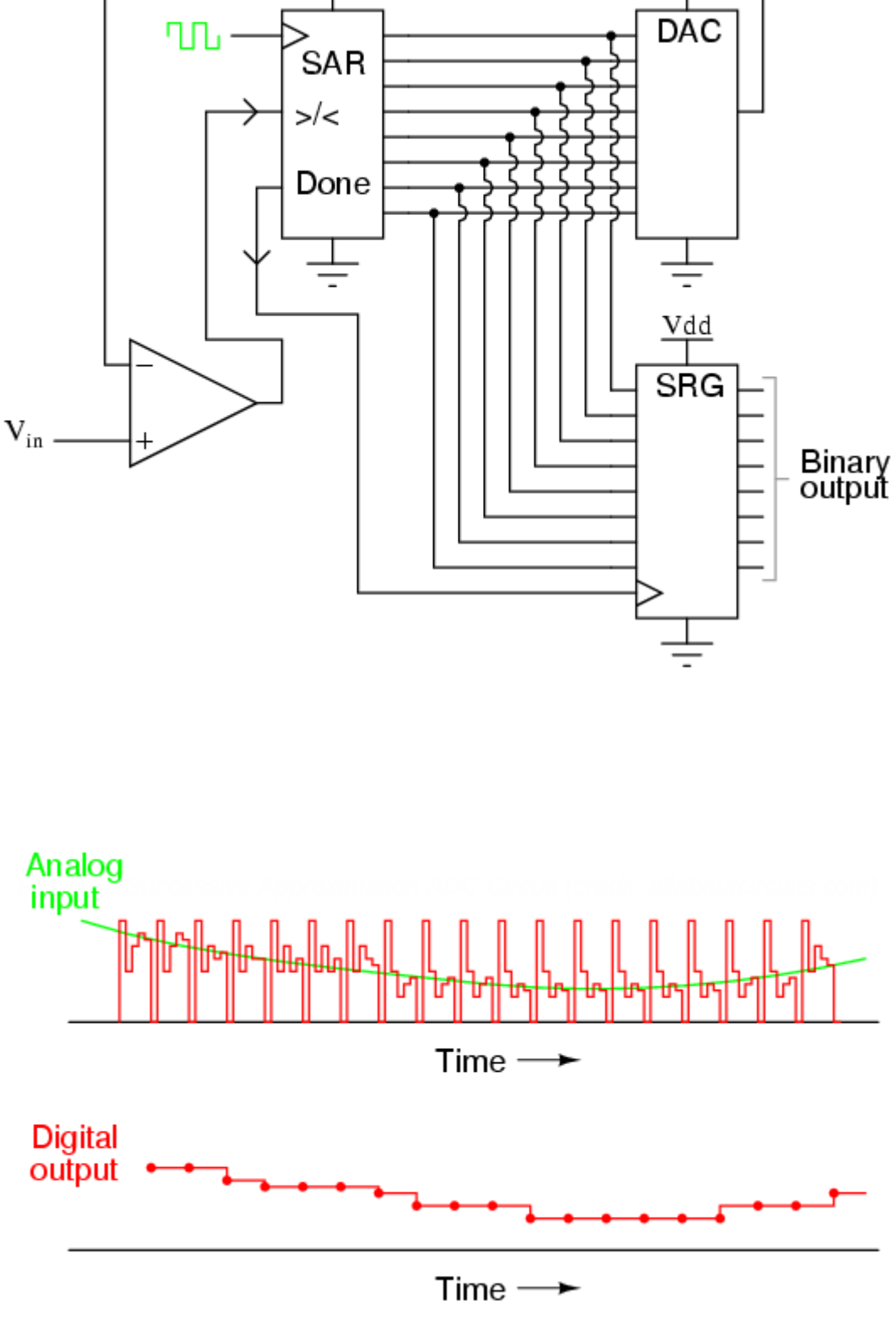


Figure 6. Successive Approximation ADC Timing (credit: allaboutcircuits.com)

Tracking ADC

A Tracking ADC works similarly to the Digital Ramp ADC, except instead of an up counter, it utilizes an

up-down counter. The output of the comparator determines whether the counter increments or decrements. It doesn't use a register to hold the processed value since it's constantly tracing the input value.

Note that this type gnal. Additionally, it suf vn as bit bobble.

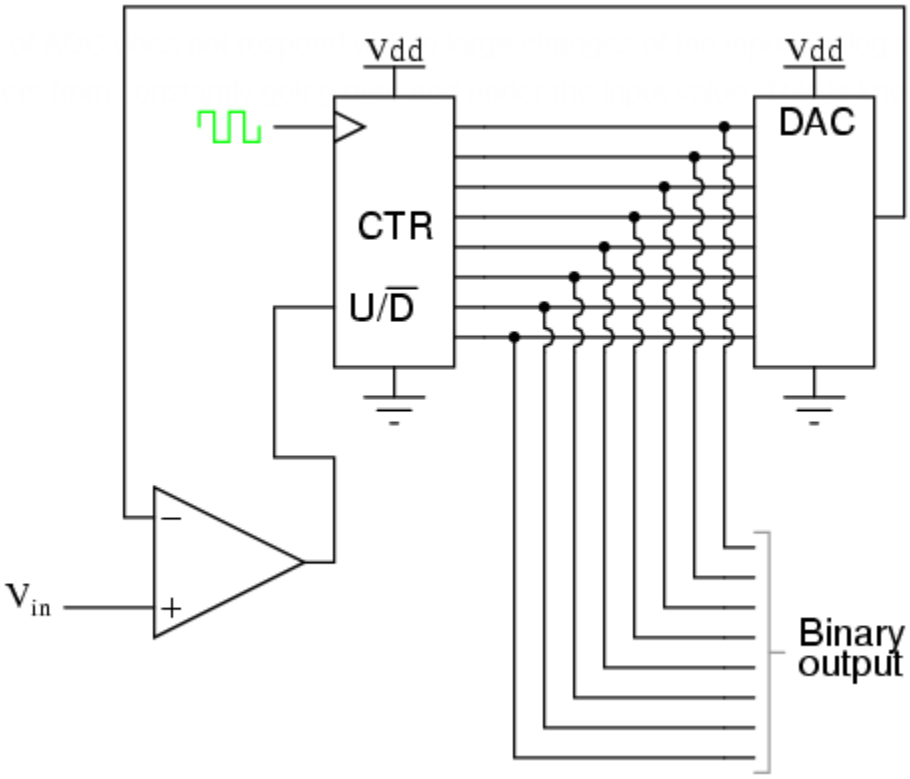


Figure 7. Tracking ADC Circuit (credit: allaboutcircuits.com)

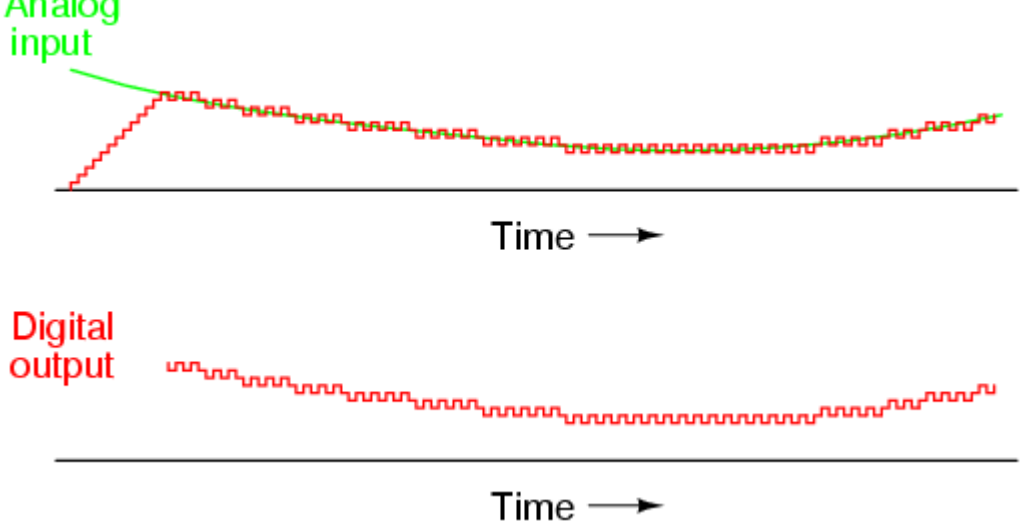


Figure 8. Tracking ADC Timing (credit: allaboutcircuits.com)

DAC-free ADCs

Besides Flash ADC, all previous ADC circuits rely on using DACs to convert an estimated digital value to an analog one and compare it to the input signal. There are other types of ADC technologies that do not use DACs. They rely on the known time it takes an RC circuit to discharge to match the input analog signal. Single Slope, Dual Slope, and Delta-Sigma ADCs implement this concept.

ADC Parameters

Resolution

This is typically the most-highlighted aspect of any ADC technology. Resolution refers to the number of bits of the ADC's output. It's a measurement of how coarse/fine the converted value is. A four bit 5V ADC offers 16 values for the voltage range 0 V to 5 V (i.e. roughly 312 mV per bit increment). A 10 bit 5V ADC offers 1024 values for the same voltage range (roughly 5 mV per bit increment).

Sampling Frequency

This is simply the circuit's latency (i.e. the rate of converting an analog input signal to digital bits). The highest frequency of an analog signal that a given ADC circuit is able to adequately capture is known as Nyquist frequency. Nyquist frequency is equal to one-half of the circuits sampling frequency. Therefore, to adequately convert an analog signal of frequency n Hz, one must have an ADC circuit with 2n Hz

sampling frequency. Otherwise, aliasing happens. Aliasing occurs when an ADC circuit samples an input signal too slowly, thus producing an output signal that is not the *true* input signal, but rather an alias of it.

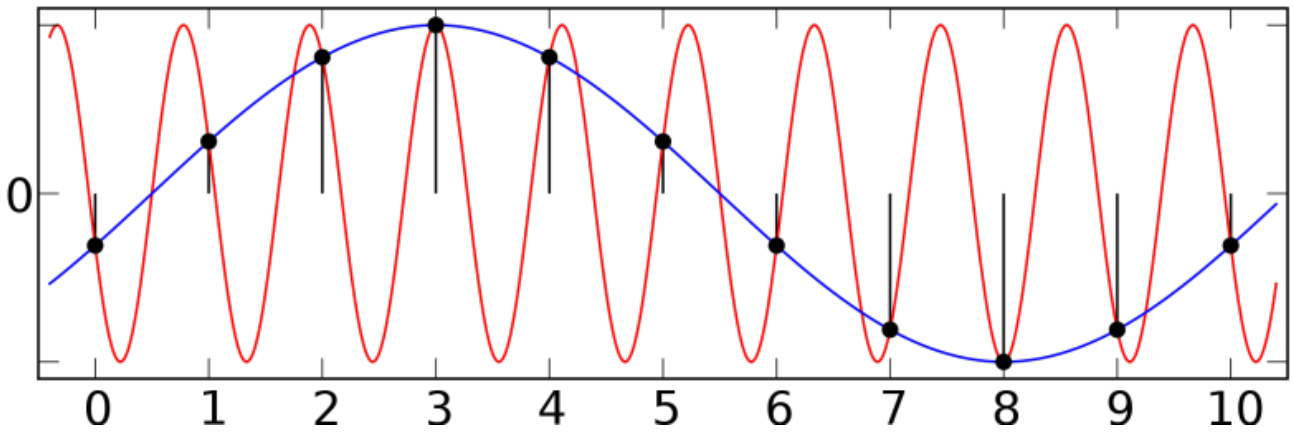


Figure 9. ADC Aliasing

Step Recovery

This is a measurement of how quickly an ADC's output is able to respond to a sudden change in input. For example, flash and successive approximation ADCs are able to adjust relatively quickly to input changes while tracking ADC struggles with large input changes.

Range

This is a measurement of the range of voltages that an ADC circuit is able to capture and output. For example, the LPC40xx has a range of 0V to 3.3V. Other ADCs may have bigger ranges or even variable ranges that a user can select, such as this device: https://www.mouser.com/ds/2/609/AD7327-EP-916882.pdf

Error

This is a measurement of the systematic error of any given ADC circuit. This is measured by comparing the actual input signal to its digital output equivalent. Note that, this error measurement is only valid within the range of the ADC in question.

ADC Driver for LPC40xx

The ADC is configured using the following registers:

1. Power: In the PCONP register (Section 3.3.2.2), set the PCADC bit.

   Remark: On reset, the ADC is disabled. To enable the ADC, first set the PCADC bit, and then enable the ADC in the AD0CR register (bit PDN Table 678). To disable the ADC, first clear the PDN bit, and then clear the PCADC bit.
2. Peripheral clock: The ADC operates from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See Section 3.3.3.5. To scale the clock for the ADC, see bits CLKDIV in Table 678.
3. Pins: Enable ADC0 pins and pin modes for the port pins with ADC0 functions through the relevant IOCON registers (Section 7.4.1).
4. Interrupts: To enable interrupts in the ADC, see Table 682. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register. Disable the ADC interrupt in the NVIC using the appropriate Interrupt Set Enable register.
5. DMA: See Section 32.6.4. For GPDMA system connections, see Table 696.

Figure 10. LPC40xx User Manual ADC Instructions

Table 677. Register overview: ADC (base address 0x4003 4000)

Generic Name	Access	Address offset	Description	Reset value[1]	Table
CR	R/W	0x000	A/D Control Register. The ADCR register must be written to select the operating mode before A/D conversion can occur.	1	678
GDR	R/W	0x004	A/D Global Data Register. This register contains the ADC's DONE bit and the result of the most recent A/D conversion.	NA	679
INTEN	R/W	0x00C	A/D Interrupt Enable Register. This register contains enable bits that allow the DONE flag of each A/D channel to be included or excluded from contributing to the generation of an A/D interrupt.	0x100	680
DR0	RO	0x010	A/D Channel 0 Data Register. This register contains the result of the most recent conversion completed on channel 0.	NA	681
DR1	RO	0x014	A/D Channel 1 Data Register. This register contains the result of the most recent conversion completed on channel 1.	NA	681
DR2	RO	0x018	A/D Channel 2 Data Register. This register contains the result of the most recent conversion completed on channel 2.	NA	681
DR3	RO	0x01C	A/D Channel 3 Data Register. This register contains the result of the most recent conversion completed on channel 3.	NA	681
DR4	RO	0x020	A/D Channel 4 Data Register. This register contains the result of the most recent conversion completed on channel 4.	NA	681
DR5	RO	0x024	A/D Channel 5 Data Register. This register contains the result of the most recent conversion completed on channel 5.	NA	681
DR6	RO	0x028	A/D Channel 6 Data Register. This register contains the result of the most recent conversion completed on channel 6.	NA	681
DR7	RO	0x2C	A/D Channel 7 Data Register. This register contains the result of the most recent conversion completed on channel 7.	NA	681
STAT	RO	0x030	A/D Status Register. This register contains DONE and OVERRUN flags for all of the A/D channels, as well as the A/D interrupt/DMA flag.	0	682
TRM	R/W	0x034	ADC trim register.	0	683

Figure 11. LPC40xx User Manual ADC Control Register

PWM (Pulse Width Modulation)

Objective

To learn about the use of PWM signals, their related parameters, and how to set up an ADC driver for the LPC40xx.

What is a PWM signal?

A Pulse Width Modulation (PWM) signal is simply a digital signal that is on (high) for part of its period and off (low) for the remainder of its period. If such a signal is on half the time and off the other half, t

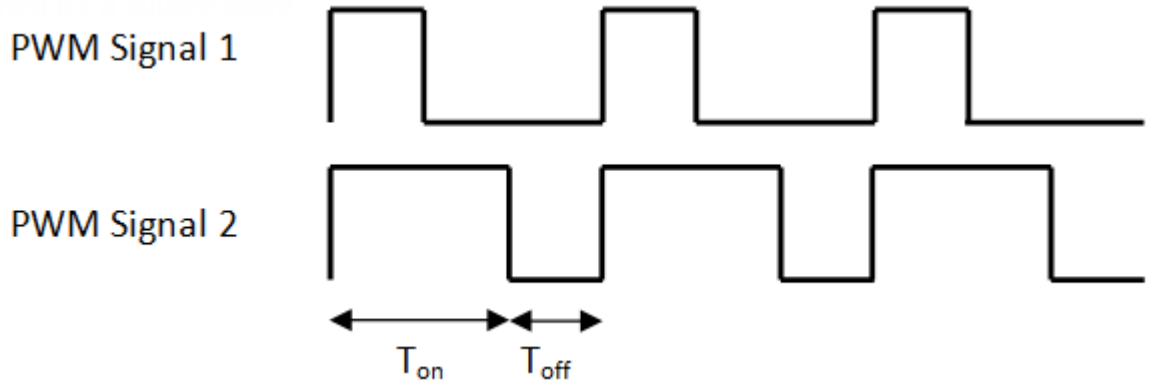


Figure 1. PWM Signal (credit: www.bvsystems.be)

PWM Parameters

Duty Cycle

A duty cycle of a certain PWM signal is given as a percentage, and it represents the ratio of the signal "on" time to the signal's full period. In other words, if the duty cycle of a signal is said to be 75%, it means that this signal is high for 75% of its period and low for the remaining 25%. 100% duty cycle implies a constantly high signal, and a 0% duty cycle implies a constantly grounded signal.

Frequency

The frequency of a PWM signal (just like any electrical signal) refers to the rate at which the signal repeats per second. A 1 Hz signal repeats every 1 second while a 1 kHz signal repeats every 1 millisecond.

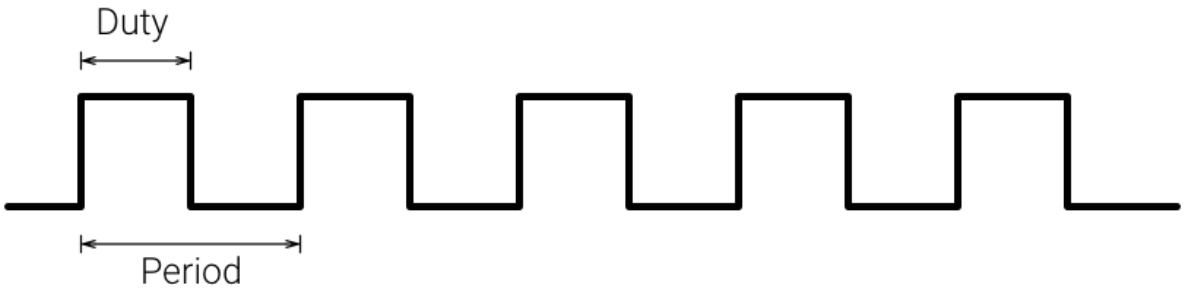


Figure 2. Parameters of a PWM signal

PWM Signal Applications

Generally speaking, a PWM signal is a way for a digital system to interface with an analog device.

DC Motors

DC Motors are controllable via a PWM signal. The duty cycle of the signal is typically linearly proportional to the velocity of the motor. For example, a 60 RPM motor driven by a 50% duty cycle PWM signal will rotate at a 30 RPM velocity. It's worth noting that such a signal needs to run at a high enough frequency (10 kHz for example) so the motor can rotate smoothly. A low-frequency PWM signal (say 10 Hz) will result in an observable choppy motor motion.

LEDs

The brightness of an LED can be controlled via a reasonably high-frequency PWM signal. A 5V 50%

PWM signal applied to an LED will have the same brightness effect as a constant 2.5V signal applied to the same LED.

Servos

Servos are typically controlled by a 50 Hz PWM signal, where the duty cycle of the signal determines the angle of the servo. Typically, the duty cycle ranges from 5% to 10%, causing the servo to rotate to its smallest and largest angles, respectively.

PWM Driver for LPC40xx

26.1 Basic configuration

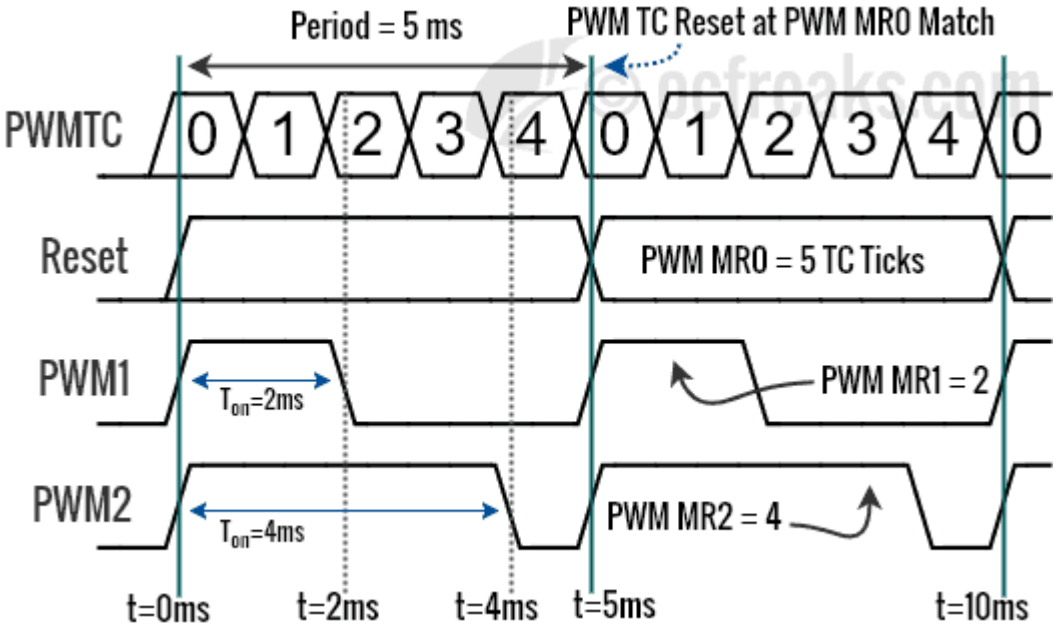
The PWM is configured using the following registers:

1. Power: In the PCONP register (Section 3.3.2.2), set bit PCPWM1.

   Remark: On reset, PWM1 is enabled (PCPWM1 = 1) and PWM0 is disabled (PCPWM1 = 0).
2. Peripheral clock: The PWMs operate from the common PCLK that clocks both the bus interface and functional portion of most APB peripherals. See Section 3.3.3.5.
3. Pins: Select PWM pins and pin modes for port pins with PWM functions through the relevant IOCON registers (Section 7.4.1).
4. Interrupts: See registers PWMMCR (Table 564) and PWMCCR (Table 567) for match and capture events. Interrupts are enabled in the NVIC using the appropriate Interrupt Set Enable register.

Theory of Operation

Behind every PWM is a Peripheral (HW) counter (TC). For "Single Edge" PWM, when the counter starts from zero, the output of the PWM (GPIO) can reset back to logical 1. Then, when the value of the "Match Register (MR)" occurs, then the PWM output can set to logical 0. Therefore, the maximum limit of the TC controls the frequency of the PWM signal, and the MR registers control the duty cycle.



Software PWM

This section demonstrates the LPC PWM operation in software. The LPC processor implements similar code, but in the hardware.

```
void lpc_pwm(void)
{
  bool GPIO_PWM1 = true; // Hypothetical GPIO that this PWM channel controls
  uint32_t TC = 0;    // Hardware counter
  uint32_t MR0 = 500; // TC resets when it matches this

  // Assumptions: SW instructions add no latency, and delay_us() is the only instruction that takes ti
  while (1)
  {
    if (++TC >= MR0) {
      TC = 0;
      GPIO_PWM1 = true;  // GPIO is HIGH on the reset of TC
    }

    if (TC >= MR1) {
      GPIO_PWM1 = false; // GPIO resets upon the match register
    }
```

```
    // 1uS * 500 = 500uS, so 2Khz PWM
    delay_us(1);
  }}
```

Registers of relevance

What you are essentially trying to control is the PWM frequency and the PWM duty cycle. For instance, a 50% duty cycle with just a 1Hz PWM will blink your LED once a second. But a 50% duty cycle 1Khz signal will dim your LED to 50% of the total brightness it is capable of. There are "rules" that the PWM module uses to alter a GPIO pin's output and **these rules are what you are trying to understand**. So read up on "Rules of Single Edge Conrolled PWM" in your datasheet and overall the LPC PWM chapter **at minimum 10 times to understand it**. You may skip the sections regarding "capture", and "interrupts". Furthermore, to use the simplified PWM, you can use the ***Single Edge*** PWM rather than the more complex ***Double Edge*** because the Single Edge edge PWM is controlled by a dedicated MR register.

TC, MR0, MCR and PR: The Prescalar (PR) register controls the tick rate of the hardware counter that can alter the **frequency** of your PWM. For instance, when the CPU clock is 10Mhz, and the PR = 9, then the TC counts up at the rate of 10/(9+1) = 1 Mhz. Hence, the PR affects the frequency, but we still need a "max count" to set the frequency with precision. So if the TC increments at 1Mhz, and MR0 is set to 100, then you will have 1000Khz/100 = 10Khz PWM.

The MCR register controls what happens to the TC when a match occurs. The one subtle, but **important** thing we need to do is that when the MR0 match occurs, we need the TC to reset to zero to be able to use MR0 as a frequency control.

TCR and PCR: The PCR register enables the channels, so if you have PWM1.4 as an output, that means you need to enable channel 4. The TCR register is a key register that will enable your PWM module.

Lab: ADC + PWM

Objective

Improve an ADC driver, and use an existing PWM driver to design and implement an embedded application, which uses RTOS queues to communicate between tasks.

This lab will utilize:

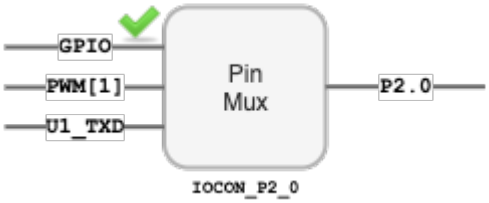
- ADC Driver
  - You will improve the driver functionality
  - You will use a potentiometer that controls the analog voltage feeding into an analog pin of your microcontroller
- PWM Driver
  - You will use an existing PWM Driver to control a GPIO
  - An led brightness will be controlled, or you can create multiple colors using an RGB LED
- FreeRTOS Tasks
  - You will use FreeRTOS queues

Assignment

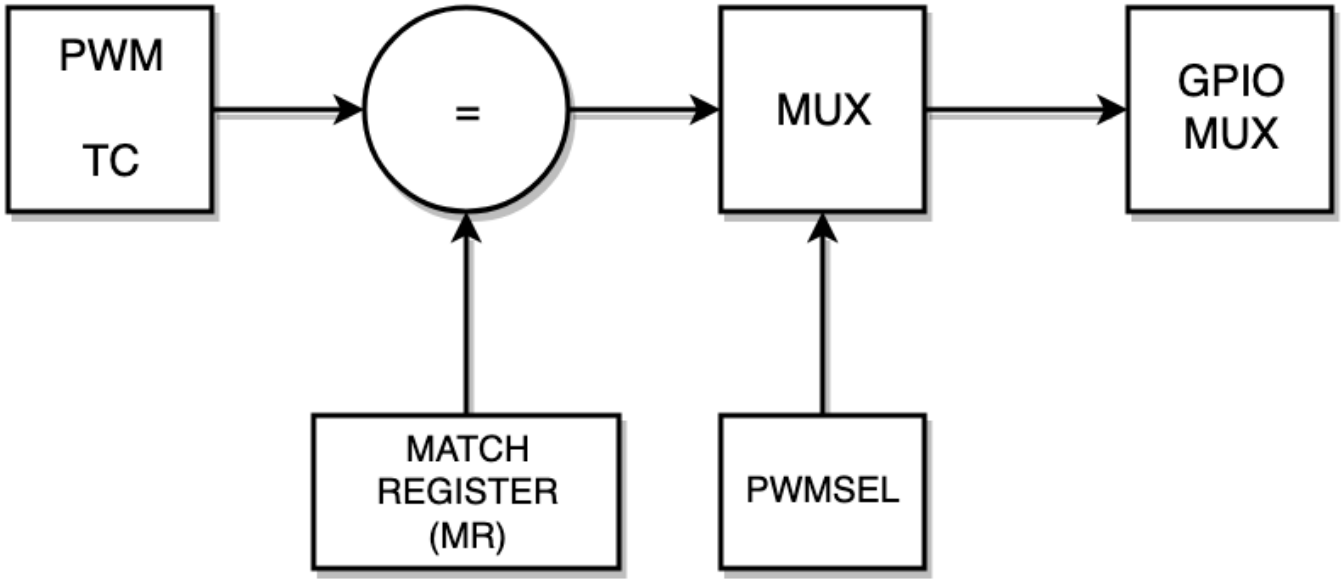
> **Preparation:**
> Before you start the assignment, please read the following in your LPC User manual (UM10562.PDF)
> - Chapter 7: I/O configuration
> - Chapter 32: ADC

Part 0: Use PWM1 driver to control a PWM output pin

The first thing to do is to select a pin to function as a PWM signal. This means that once you select a pin function correctly, then the pin's function is controlled by the PWM peripheral and you cannot control the pin's HIGH or LOW using the GPIO peripheral. By default, a pin's function is as GPIO, but for example, you can disconnect this function and select the PWM function by using the `IOCON_P2_0`



1. Re-use the PWM driver
   - Study the `pwm1.h` and `pwm1.c` files under `l3_drivers` directory
2. Locate the pins that the PWM peripheral can control at `Table 84: FUNC values and pin functions`
   - These are labeled as `PWM1[x]` where `PWM1` is the peripheral, and `[x]` is a channel
     - So `PWM1[2]` means PWM1, channel 2
   - Now find which of these channels are available as a free pin on your SJ2 board and connect the RGB led
     - Set the `FUNC` of the pin to use this GPIO as a PWM output
3. Initialize and use the PWM-1 driver
   - Initialize the PWM1 driver at a frequency of your choice (greater than 30Hz for human eyes)
   - Set the duty cycle and let the hardware do its job :)
4. You are finished with Part 0 if you can demonstrate control over an LED's brightness using the HW based PWM method




```
#include "pwm1.h"
#include "FreeRTOS.h"
#include "task.h"
void pwm_task(void *p) {
  pwm1__init_single_edge(1000);

  // Locate a GPIO pin that a PWM channel will control
  // NOTE You can use gpio__construct_with_function() API from gpio.h
  // TODO Write this function yourself
  pin_configure_pwm_channel_as_io_pin();

  // We only need to set PWM configuration once, and the HW will drive
  // the GPIO at 1000Hz, and control set its duty cycle to 50%
  pwm1__set_duty_cycle(PWM1__2_0, 50);

  // Continue to vary the duty cycle in the loop
  uint8_t percent = 0;
  while (1) {
    pwm1__set_duty_cycle(PWM1__2_0, percent);

    if (++percent > 100) {
      percent = 0;
```

```
    }

    vTaskDelay(100);
  }
}
void main(void) {
  xTaskCreate(pwm_task, ...);
  vTaskStartScheduler();}
```

Part 1: Alter the ADC driver to enable `Burst Mode`

- Study `adc.h` and `adc.c` files in `l3_drivers` directory and correlate the code with the ADC peripheral by reading the LPC User Manual.
  - **Do not skim over the driver, make sure you fully understand it.**
- Identify a pin on the SJ2 board that is an ADC channel going into your ADC peripheral.
  - Reference the I/O pin map section in `Table 84,85,86: FUNC values and pin functions`
- Connect a potentiometer to one of the ADC pins available on SJ2 board. Use the ADC driver and implement a simple task to decode the potentiometer values and print them. Values printed should range from 0-4095 for different positions of the potentiometer.

```
// TODO: Open up existing adc.h file
// TODO: Add the following API
/**
 * Implement a new function called adc__enable_burst_mode() which will
 * set the relevant bits in Control Register (CR) to enable burst mode.
 */
void adc__enable_burst_mode(void);
/**
 * Note:
 * The existing ADC driver is designed to work for non-burst mode
 *
 * You will need to write a routine that reads data while the ADC is in burst mode
 * Note that in burst mode, you will NOT read the result from the GDR register
```

```
 * Read the LPC user manual for more details
*/uint16_t adc__get_channel_reading_with_burst_mode(uint8_t channel_number);
```

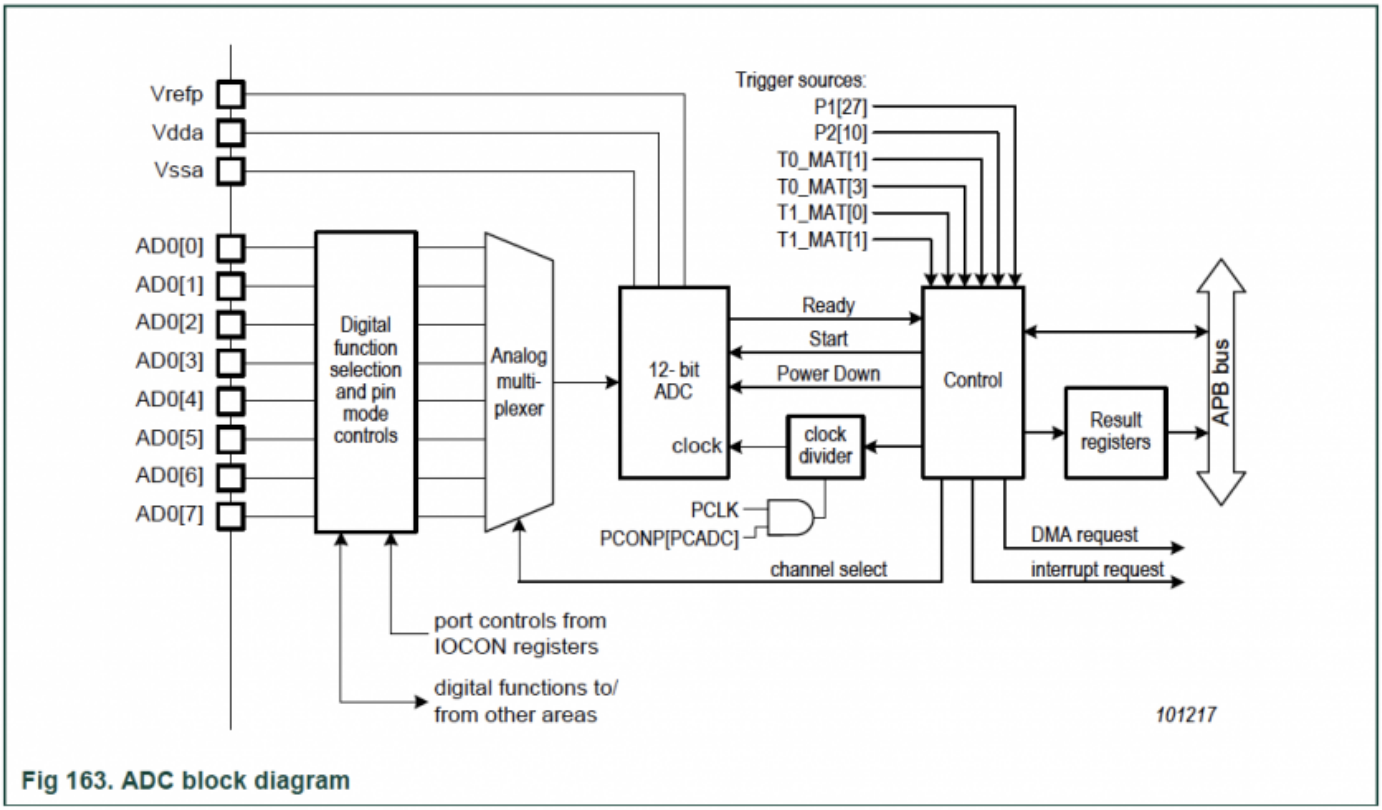


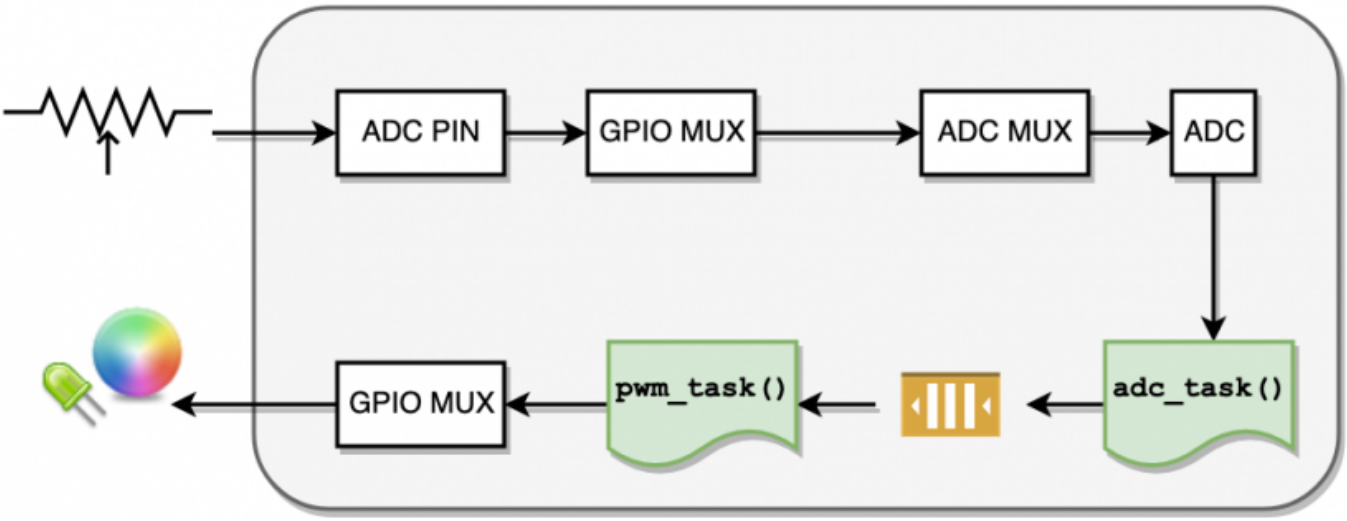
Fig 163. ADC block diagram

```
#include "adc.h"

#include "FreeRTOS.h"
#include "task.h"
void adc_pin_initialize(void) {
  // TODO: Ensure that you also set ADMODE to 0
  // TODO: Ensure you set pull/up and pull/down bits 0
  // TODO: Then use gpio__construct_with_function(...)
}
void adc_task(void *p) {
  adc_pin_initialize();
  adc__initialize();

  // TODO This is the function you need to add to adc.h
  // You can configure burst mode for just the channel you are using
```

```
  adc__enable_burst_mode();

  // Configure a pin, such as P1.31 with FUNC 011 to route this pin as ADC channel 5
  // You can use gpio__construct_with_function() API from gpio.h
  pin_configure_adc_channel_as_io_pin(); // TODO You need to write this function

  while (1) {
    // Get the ADC reading using a new routine you created to read an ADC burst reading
    // TODO: You need to write the implementation of this function
    const uint16_t adc_value = adc__get_channel_reading_with_burst_mode(ADC__CHANNEL_2);

    vTaskDelay(100);
  }
}
void main(void) {
  xTaskCreate(adc_task, ...);
  vTaskStartScheduler();}
```

Part 2: Use FreeRTOS Queues to communicate between tasks

- Read this chapter to understand how FreeRTOS queues work
- Send data from the `adc_task` to the RTOS queue
- Receive data from the queue in the `pwm_task`




```
#include "adc.h"

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
// This is the queue handle we will need for the xQueue Send/Receive API
static QueueHandle_t adc_to_pwm_task_queue;
void adc_task(void *p) {
  // NOTE: Reuse the code from Part 1

  int adc_reading = 0; // Note that this 'adc_reading' is not the same variable as the one from adc_ta
  while (1) {
    // Implement code to send potentiometer value on the queue
    // a) read ADC input to 'int adc_reading'
    // b) Send to queue: xQueueSend(adc_to_pwm_task_queue, &adc_reading, 0);
    vTaskDelay(100);
  }
}
void pwm_task(void *p) {
  // NOTE: Reuse the code from Part 0
  int adc_reading = 0;
  while (1) {
    // Implement code to receive potentiometer value from queue
    if (xQueueReceive(adc_to_pwm_task_queue, &adc_reading, 100)) {
```

```
    }

    // We do not need task delay because our queue API will put task to sleep when there is no data in
    // vTaskDelay(100);
  }
}
void main(void) {
  // Queue will only hold 1 integer
  adc_to_pwm_task_queue = xQueueCreate(1, sizeof(int));
  xTaskCreate(adc_task, ...);
  xTaskCreate(pwm_task, ...);
  vTaskStartScheduler();}
```

Part 3: Allow the Potentiometer to control the RGB LED

At this point, you should have the following structure in place:

- ADC task is reading the potentiometer ADC channel, and sending its values over to a queue
- PWM task is reading from the queue

Your next step is:

- PWM task should read the ADC queue value, and control the an LED

Final Requirements

Minimal requirement is to use a single potentiometer, and vary the light output of an LED using a PWM. For **extra credit**, you may use 3 PWM pins to control an RGB led and create color combinations using a single potentiometer.

- Make sure your **Part 3** requirements are completed
- `pwm_task` should print the values of MR0, and the match register used to alter the PWM LEDs

    - For example, MR1 may be used to control P2.0, so you will print MR0, and MR1
    - Use memory mapped `LPC_PWM` registers from `lpc40xx.h`
- Make sure **BURST MODE** is enabled correctly.
- `adc_task` should convert the digital value to a voltage value (such as 1.653 volts) and print it out to the serial console
    - Remember that your VREF for ADC is 3.3, and you can use ratio to find the voltage value
    - `adc_voltage / 3.3 = adc_reading / 4095`