General Purpose I/O

Overview

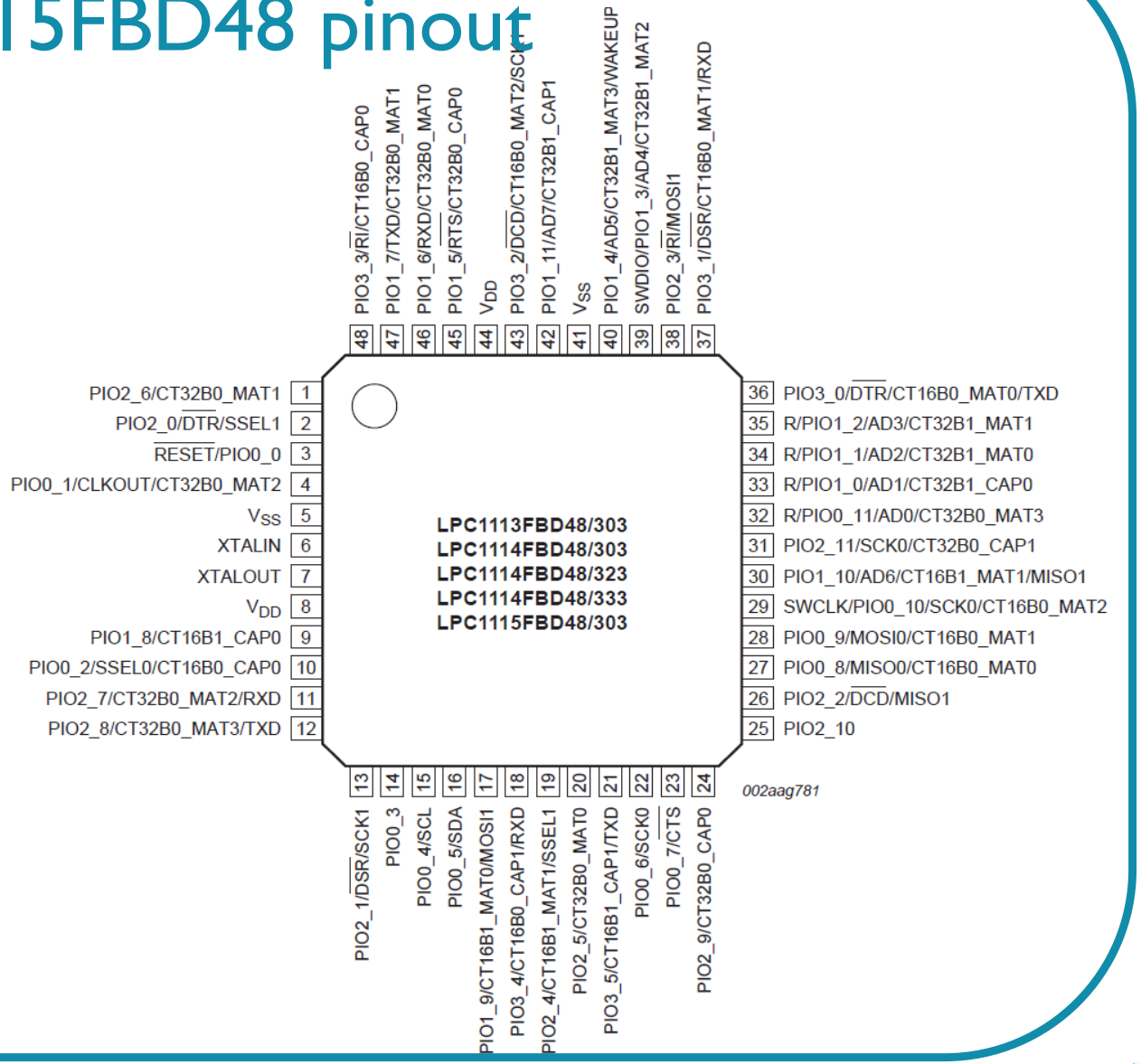
- How do we make a program light up LEDs in response to a switch?
- GPIO
  - Basic Concepts
  - Port Circuitry
  - Control Registers
  - Accessing Hardware Registers in C
- Circuit Interfacing
  - Inputs
  - Outputs
- Additional Configuration
  - GPIO as Interrupt

Basic Concepts

- GPIO = General-purpose input and output (digital)
  - Input: program can determine if input signal is a 1 or a 0
  - Output: program can set output to 1 or 0
- Can use this to interface with external devices or on board peripherals
  - Input: switch, button……
  - Output: LEDs, speaker……

LPC1115FBD48 pinout

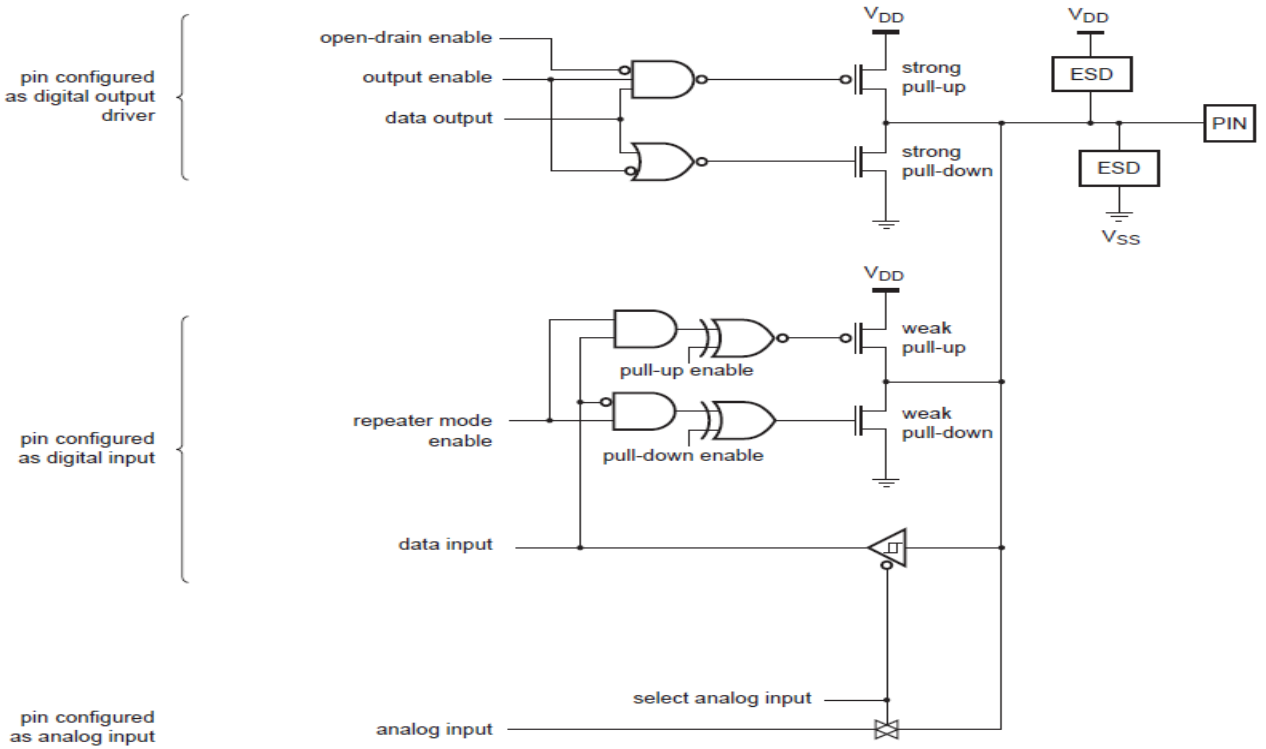
- Port 0(PIO0) through Port 3(PIO3)
- Not all port bits are available on the board
- Quantity depends on package pin count and MCU layout



ARM

GPIO Port Bit Circuitry in MCU

- Configuration
  - Direction
  - Interrupt
  - Modes
  - Mux
  - Edge or level-sensitive(high-active or low-active) interrupt request
- Data
  - Output
  - Input
  - Analogue



Open-drain mode available on series LPC1100L and LPC1100XL.

ARM

Control Registers

- Each general-purpose I/O port has
  - 32-bit configuration register GPIOnDIR (Direction register)
  - four 32-bit interrupt configuration registers
    - GPIOnIS (Interrupt sense register)
    - GPIOnIBE (Interrupt both edges register)
    - GPIOnIEV (Interrupt event register)
    - GPIOnIE (Interrupt mask register)
  - two 32-bit interrupt status registers
    - GPIOnRIS (Raw Interrupt status register)
    - GPIOnMIS (Masked interrupt status register)
  - 32-bit Interrupt clear register (GPIOnIC,write only)
  - two 32-bit data registers GPIOnData(Data register and data address masking register)

GPIO Data Direction Register

- Each bit can be configured differently
- Reset clears port bit direction to 0 (input)
- Despite the fact that registers are 32-bit, most bits are reserved (31:12)

- 0 = Pin PIOn_x is configured as input.
- 1 = Pin PIOn_x is configured as output.

E.g. to set Port0 Pin0 as output, set the first bit of GPIO0DIR register.

GPIO Interrupt Registers

- GPIO can be configured as external interrupt
- ISENSE (Interrupt sense register)
  - 0 = Interrupt on pin PIOn_x is configured as edge sensitive.
  - 1 = Interrupt on pin PIOn_x is configured as level sensitive.
- IBE (Interrupt both edges register)
  - 0 = Interrupt on pin PIOn_x is controlled through register GPIOnIEV.
  - 1 = Both edges on pin PIOn_x trigger an interrupt.
- IEV (Interrupt event register)
  - 0 = falling edges or LOW level on trigger an interrupt.
  - 1 = rising edges or HIGH level on trigger an interrupt.
- MASK (Interrupt mask register)
  - 0 = Interrupt on pin PIOn_x is masked.
  - 1 = Interrupt on pin PIOn_x is not masked.

GPIO Interrupt Status\Clear Registers

- RAWST (Raw interrupt status)-Read only
  - 0 = No interrupt on pin PIOn_x.
  - 1 = Interrupt requirements met on PIOn_x.
- MASK (Masked interrupt status register)-Read only
  - 0 = No interrupt or interrupt masked on pin PIOn_x.
  - 1 = Interrupt on PIOn_x.

- CLR (Interrupt clear register)-Write only
  - 0 = No effect.
  - 1 = Clears edge detection logic for pin PIOn_x.MASK (Interrupt mask register)
  - The synchronizer between the GPIO and the NVIC blocks causes a delay of 2 clocks. It is recommended to add two NOPs after the clear of the interrupt edgedetection logic before the exit of the interrupt service routine.

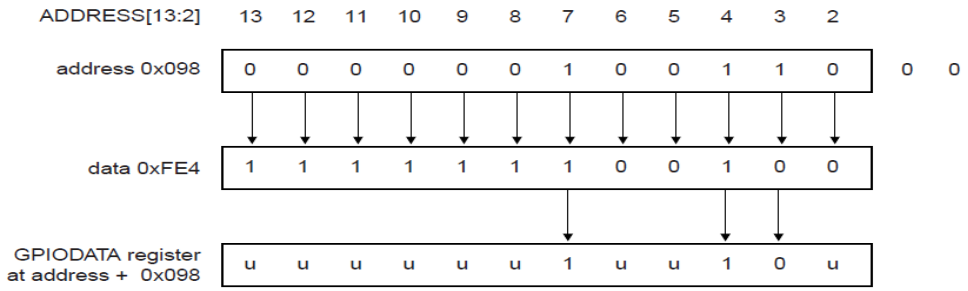
GPIO Data Registers

- Generally speaking, the GPIOnDATA register holds the current logic state of the pin (HIGH or LOW),independently of whether the pin is configured as an GPIO input or output.
- If the pin is configured as GPIO output, the current value of the GPIOnDATA register is driven to the pin.
- A read returns the current value (or the output latch)
- A write has different effects:
  - If a pin is configured as GPIO input, no effect.
  - If a pin is configured as GPIO output, the value will be driven to the pin.

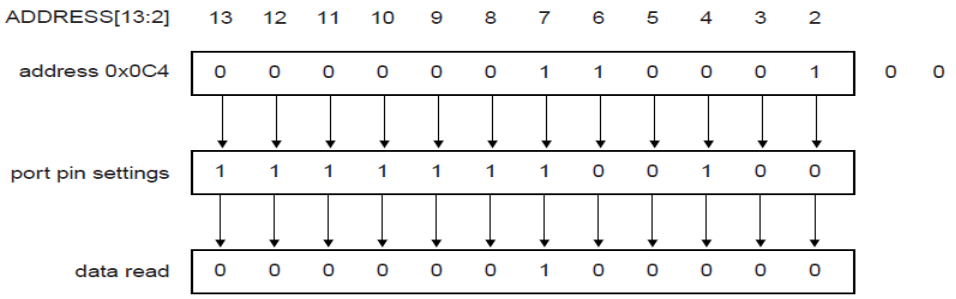
- The following rules apply when the pins are switched from input to output:
  - Pin is configured as input with a HIGH level applied:Change pin to output: pin drives HIGH level.
  - Pin is configured as input with a LOW level applied:Change pin to output: pin drives LOW level.
- Floating pins may drive an unpredictable level when switched from input to output.

Write/Read Data Operation

- Data address masking register can be used to mask certain bits so only GPIOnDATA bits masked by 1 are affected by read and write operations.
- Write Operation
  - If address bit(i+2) is HIGH, then the value can be updated
  - Otherwise, the corresponding GPIODATA register bit is left unchanged.



- Read Operation
  - Reading yields the state of port pins 11:0 ANDed with address bits 13:2.



ARM

CMSIS - Accessing Hardware Registers in C

- Header file LPC11xx.h defines C data structure types to represent hardware registers in MCU with CMSIS-Core hardware abstraction layer

```
/*------------- General Purpose Input/Output (GPIO) ------------------------*/
/** @addtogroup LPC11xx_GPIO LPC11xx General Purpose Input/Output
  @{
*/
typedef struct
{
  union {
    __IO uint32_t MASKED_ACCESS[4096];  /*!< Offset: 0x0000 (R/W) Port data Register for pins PIOn_0 to PIOn_11 */
    struct {
         uint32_t RESERVED0[4095];
    __IO uint32_t DATA;                 /*!< Offset: 0x3FFC (R/W) Port data Register */
    };
  };
       uint32_t RESERVED1[4096];
  __IO uint32_t DIR;                    /*!< Offset: 0x8000 (R/W)  Data direction Register */
  __IO uint32_t IS;                     /*!< Offset: 0x8004 (R/W)  Interrupt sense Register */
  __IO uint32_t IBE;                    /*!< Offset: 0x8008 (R/W)  Interrupt both edges Register */
  __IO uint32_t IEV;                    /*!< Offset: 0x800C (R/W)  Interrupt event Register */
  __IO uint32_t IE;                     /*!< Offset: 0x8010 (R/W)  Interrupt mask Register */
  __I  uint32_t RIS;                    /*!< Offset: 0x8014 (R/ )  Raw interrupt status Register */
  __I  uint32_t MIS;                    /*!< Offset: 0x8018 (R/ )  Masked interrupt status Register */
  __O  uint32_t IC;                     /*!< Offset: 0x801C ( /W)  Interrupt clear Register */
} LPC_GPIO_TypeDef;
/*@}*/ /* end of group LPC11xx_GPIO */
```


ARM

CMSIS C Support

- Header file lpc11xx.h also defines to LPC_GPIO_TypeDef structures

```
#define LPC_GPIO_BASE          (LPC_AHB_BASE  + 0x00000)
#define LPC_GPIO0_BASE         (LPC_AHB_BASE  + 0x00000)
#define LPC_GPIO1_BASE         (LPC_AHB_BASE  + 0x10000)
#define LPC_GPIO2_BASE         (LPC_AHB_BASE  + 0x20000)
#define LPC_GPIO3_BASE         (LPC_AHB_BASE  + 0x30000)
.....
#define LPC_GPIO0 ((LPC_GPIO_TypeDef   *)LPC_GPIO0_BASE )
#define LPC_GPIO1 ((LPC_GPIO_TypeDef   *)LPC_GPIO1_BASE )
#define LPC_GPIO2 ((LPC_GPIO_TypeDef   *)LPC_GPIO2_BASE )
#define LPC_GPIO3 ((LPC_GPIO_TypeDef   *)LPC_GPIO3_BASE )
```

Clocking Logic

- Need to enable clock to GPIO module
- By default, GPIO modules are disabled to save power
- Writing to an unclocked module triggers a hardware fault!
- Writing to system AHB clock control register (SYSAHBCLKCTRL, address 0x4004 8080) clocks to GPIO ports
- Set the bit 6 of the SYSAHBCLKCTRL

```
LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6);
```

Initializing GPIO

- Enable clock for GPIO
- Set the Pin Function , Mode , .. etc.
- Not all of these are necessary, default setting is ok (usually all bits cleared after reset)
- Simple example for initializing the orange led on the board
  - PIO0_7

```
void Init_LED(void){

    //Enable AHB clock to the GPIO domain
    LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6);
    //Set the direction for GPIO
    LPC_GPIO0->DIR |= MASK(7);
}
```

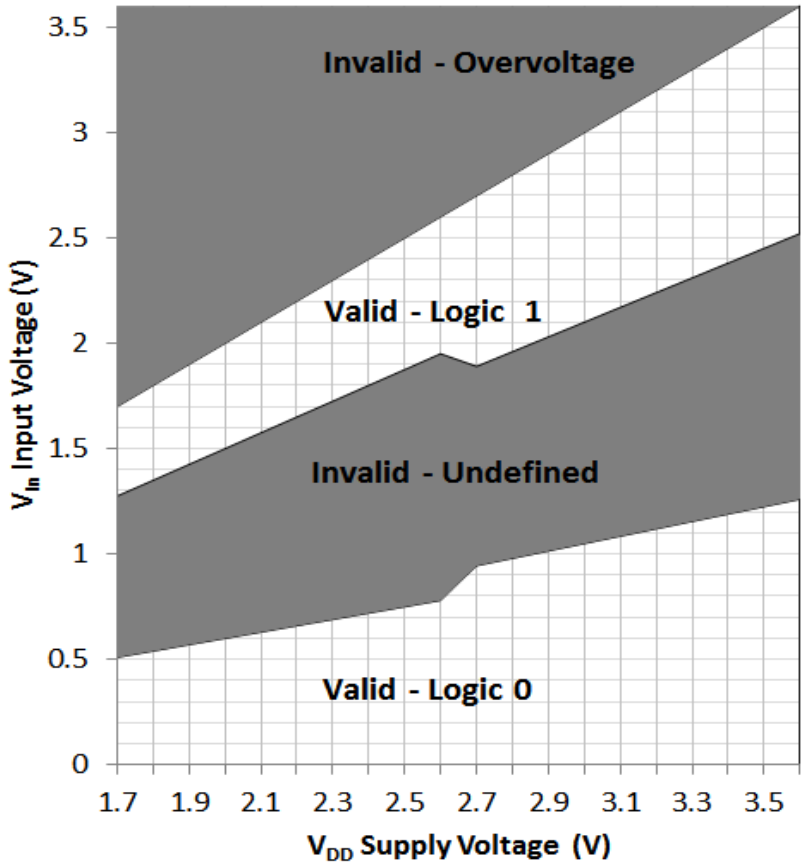
Writing/Reading Output/Input Port Data

- Write to GPIO
  - Write to the data register
  - GPIO3->DATA|=(1<<2);
  - Output High to PIO3_2
- Read from GPIO
  - Read from the data register
  - data=GPIO3->DATA&(1<<12)

Interfacing

Inputs: What's a One? A Zero?

- Input signal's value is determined by voltage
- Input threshold voltages depend on supply voltage V_{DD}
- Exceeding V_{DD} or GND may damage chip



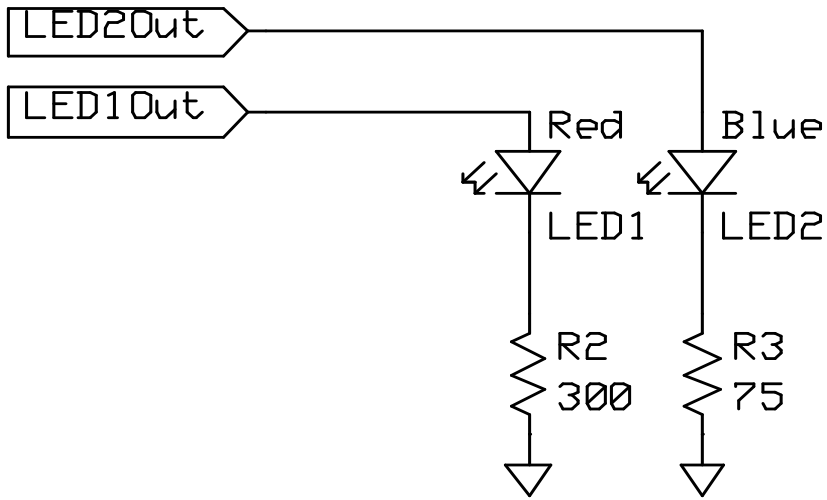
ARM

Outputs: What's a One? A Zero?

- Nominal output voltages
  - 1: V_{DD}-0.5 V to V_{DD}
  - 0: 0 to 0.5 V
- Note: Output voltage depends on current drawn by load on pin
  - Need to consider source-to-drain resistance in the transistor

Driving External LEDs

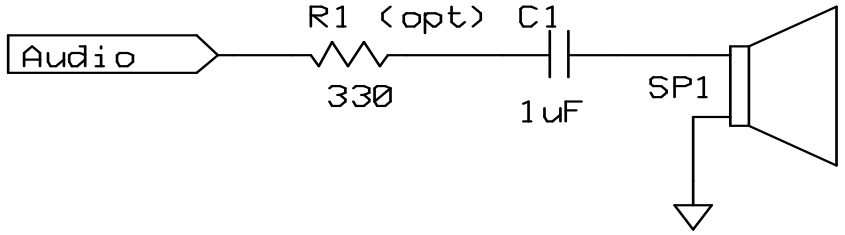
- Need to limit current to a value which is safe for both LED and MCU port driver
- Use current-limiting resistor
  - $R = (V_{DD} - V_{LED})/I_{LED}$
- Set I_{LED} = 4 mA
- V_{LED} depends on type of LED (mainly color)
  - Red: ~1.8V
  - Blue: ~2.7 V
- Solve for R given VDD = ~3.0 V
  - Red: 300 Ω
  - Blue: 75 Ω



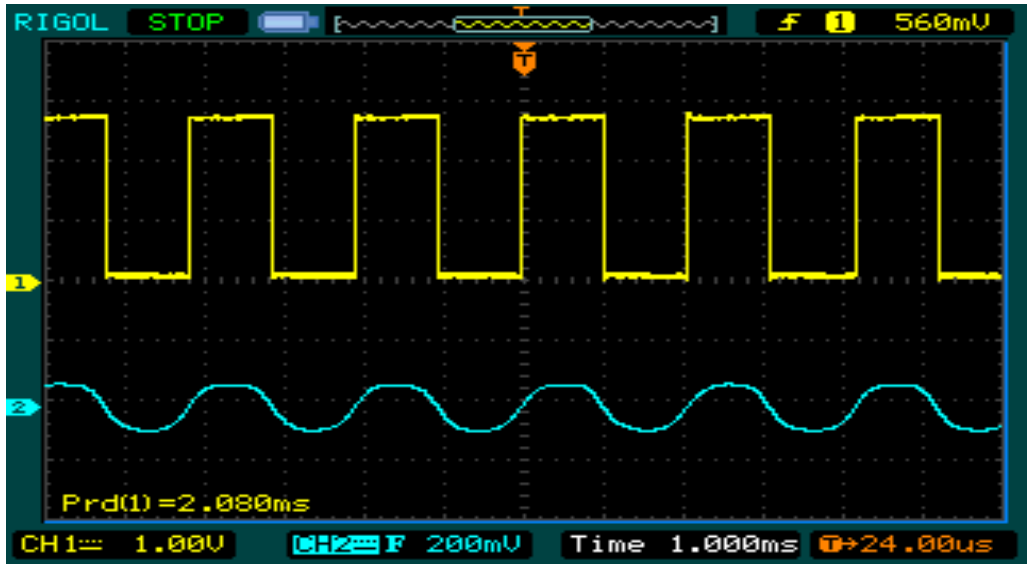
ARM

Output Example: Driving a Speaker

- Create a square wave with a GPIO output
- Use capacitor to block DC value
- Use resistor to reduce volume if needed




```
void Speaker_Beep(uint32_t frequency){
    Init_Speaker();
    while(1){
                GPIOD->BSRRL=(MASK(2));
                Delay(frequency);
                GPIOD->BSRRH=(MASK(2));
                Delay(frequency);
    }
}
```



ARM

Analog Interfacing

Why Analog

- Embedded systems often need to measure values of physical parameters
- These parameters are usually continuous (*analog*) and not in a digital form which computers (which operate on discrete data values) can process

- Temperature
  - Thermometer (do you have a fever?)
  - Thermostat for building, fridge, freezer
  - Car engine controller
  - Chemical reaction monitor
  - Safety (e.g. microprocessor processor thermal management)
- Light (or infrared or ultraviolet) intensity
  - Digital camera
  - IR remote control receiver
  - Tanning bed
  - UV monitor
- Rotary position
  - Wind gauge
  - Knobs
- Pressure
  - Blood pressure monitor
  - Altimeter
  - Car engine controller
  - Scuba dive computer
  - Tsunami detector
- Acceleration
  - Air bag controller
  - Vehicle stability
  - Video game remote
- Mechanical strain
- Other
  - Touch screen controller
  - EKG, EEG
  - Breathalyzer

ADC - The Big Picture

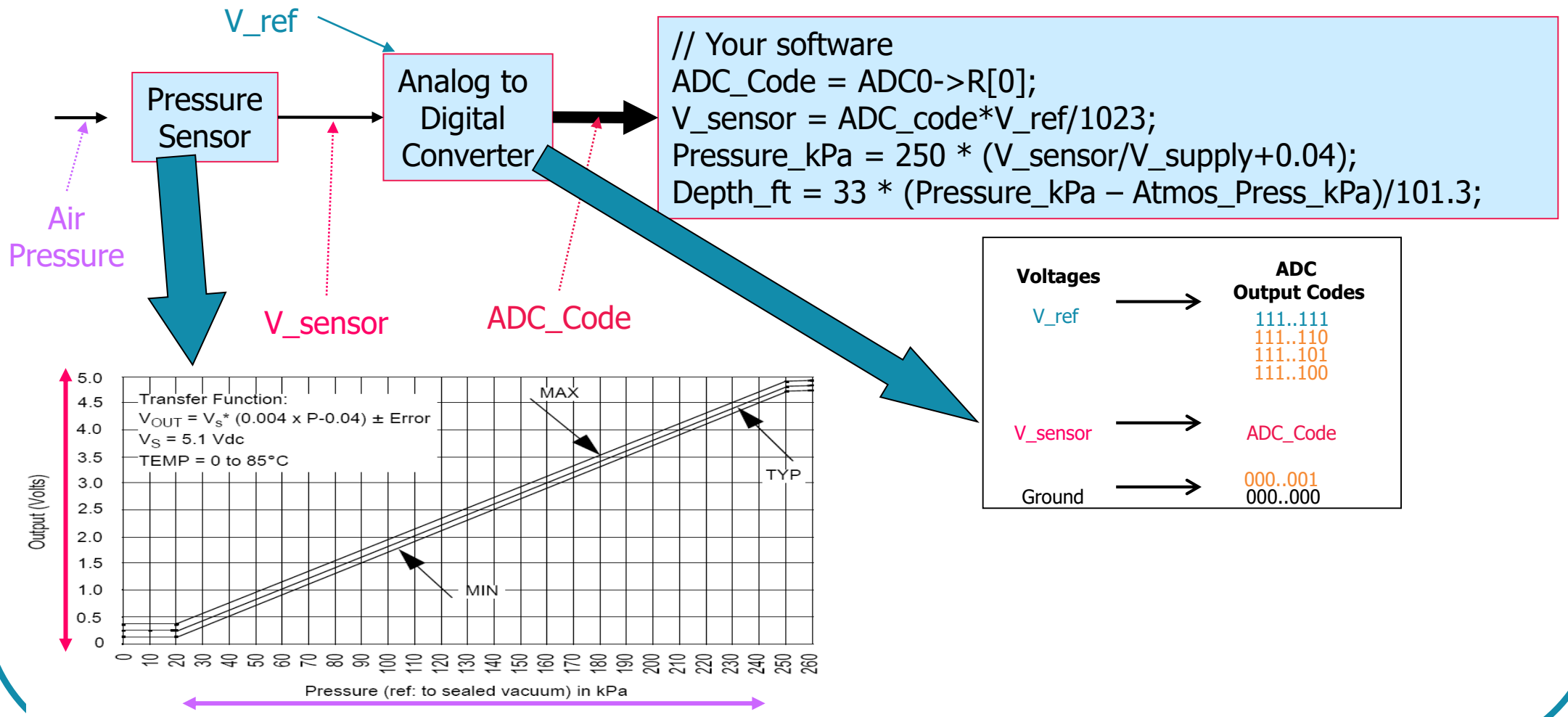
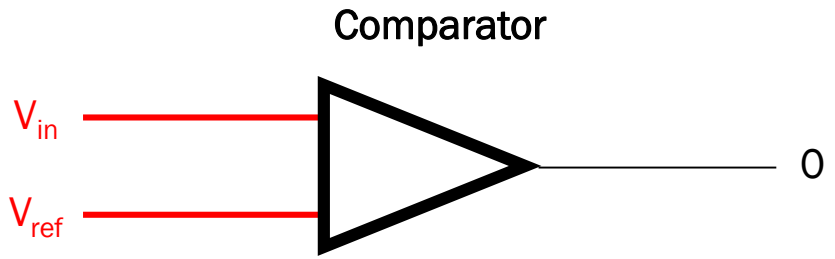


Figure 4. Output vs. Absolute Pressure

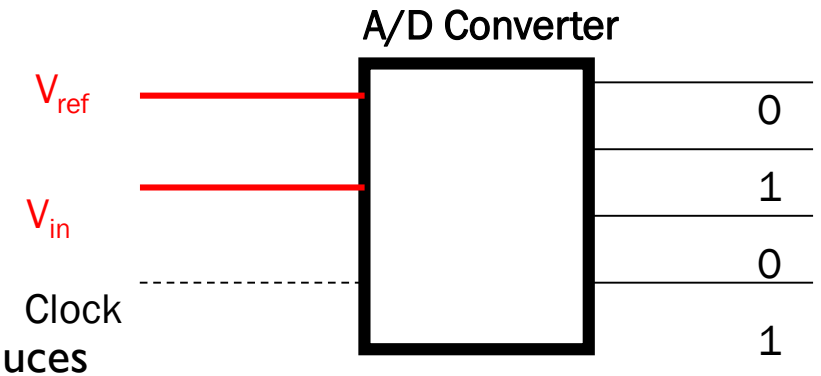
ARM

Getting From Analog to Digital

- A **Comparator** tells us "Is $V_{in} > V_{ref}$?"
  - Compares an **analog input voltage** with an **analog reference voltage** and determines which is larger, returning a 1-bit number
  - E.g. Indicate if depth > 100 ft
  - Set V_{ref} to voltage pressure sensor returns with 100 ft depth.



An **Analog to Digital converter** [AD or ADC] tells us how large V_{in} is as a fraction of V_{ref}.

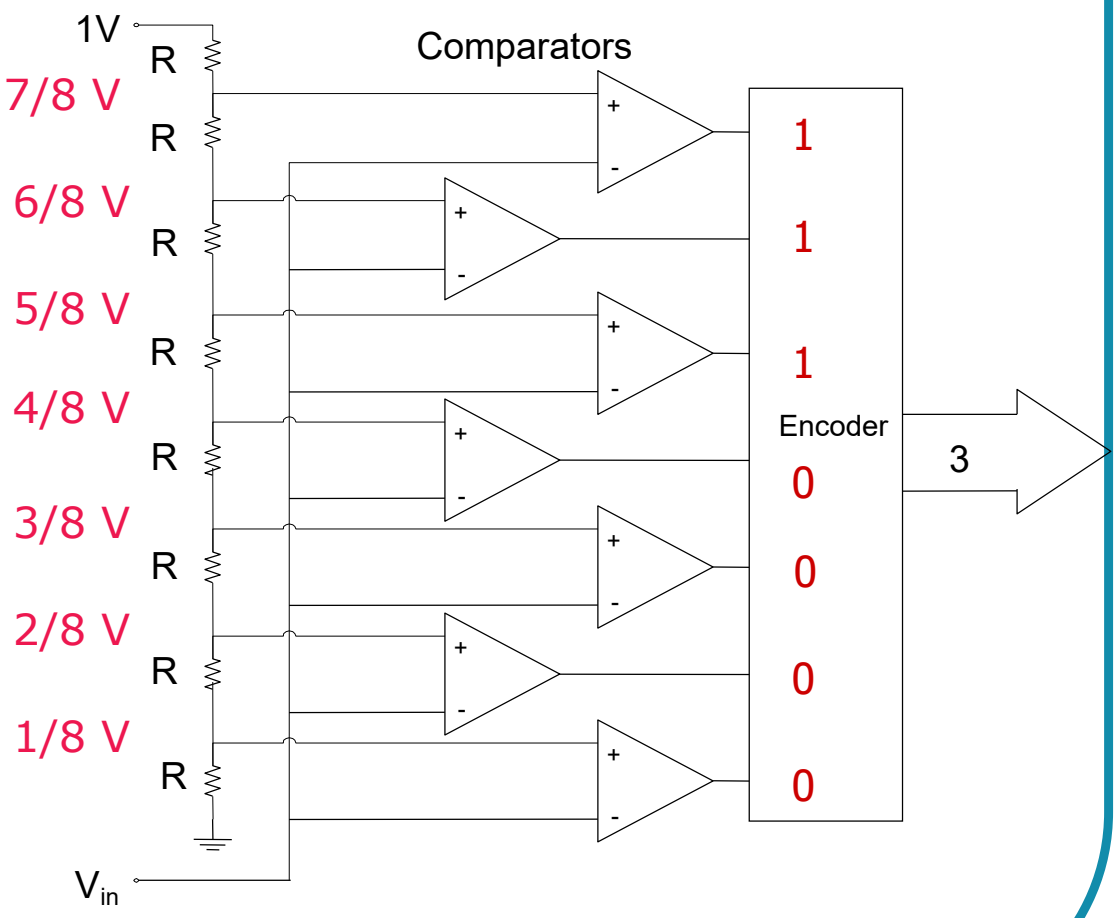


  - Reads an analog input signal (usually a voltage) and produces a corresponding multi-bit number at the output.
  - E.g. calculate the depth

ARM

ADC – Flash Conversion

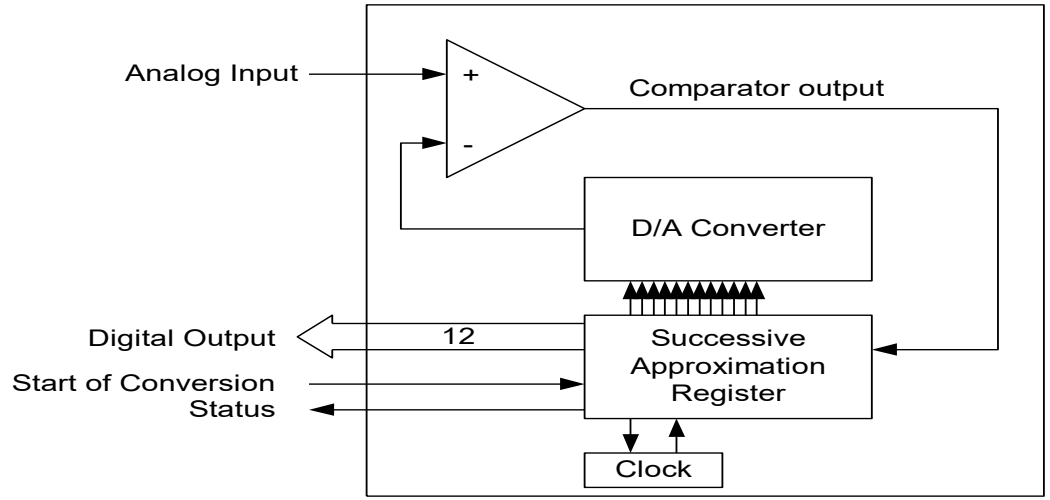
- A multi-level voltage divider is used to set voltage levels over the complete range of conversion.
- A comparator is used at each level to determine whether the voltage is lower or higher than the level.
- The series of comparator outputs are encoded to a binary number in digital logic (a priority encoder)
- Components used
  - 2^N resistors
  - 2^N-1 comparators
- Note
  - This particular resistor divider generates voltages which are *not* offset by ½ bit, so maximum error is 1 bit
  - We could change this offset voltage by using resistors of values R, 2R, 2R ... 2R, 3R (starting at bottom)



ARM

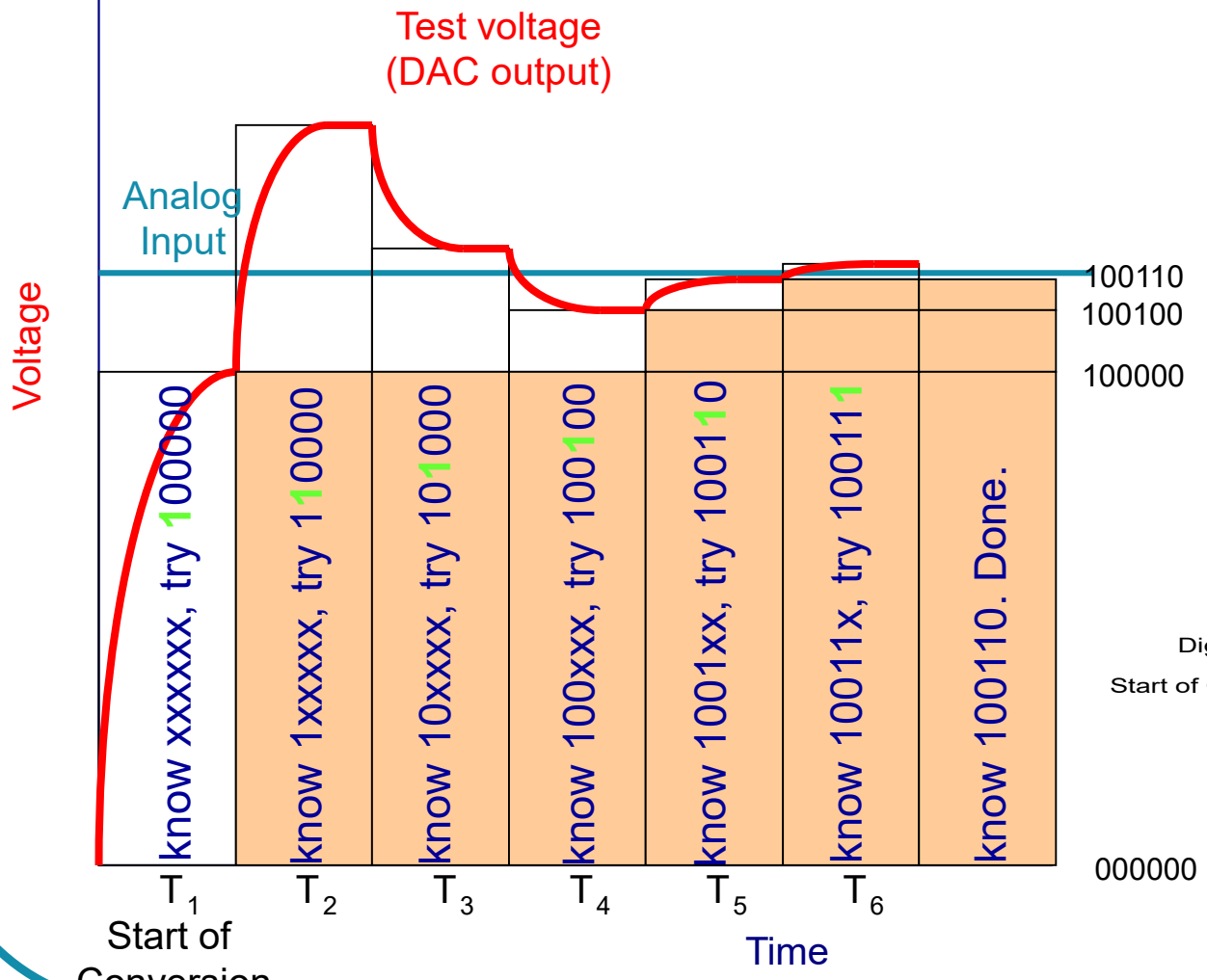
ADC - Successive Approximation Conversion

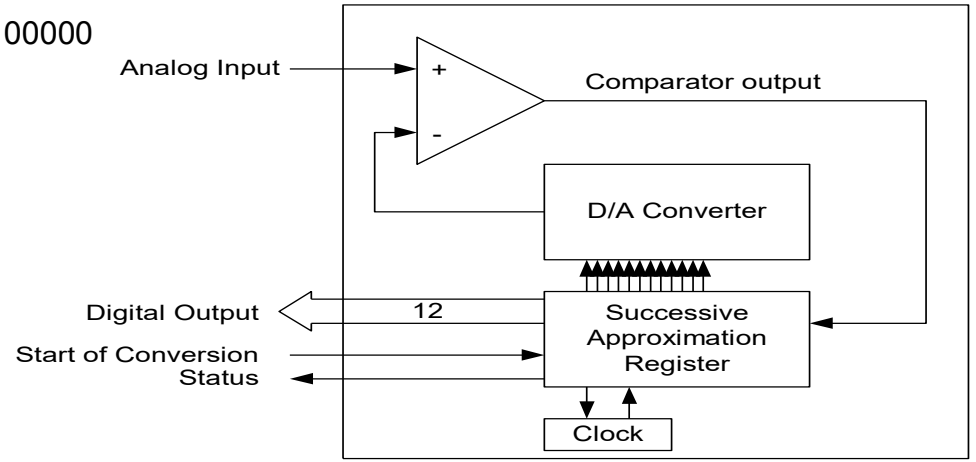
- Successively approximate input voltage by using a binary search and a DAC
- SA Register holds current approximation of result
- Set all DAC input bits to 0
- Start with DAC's most significant bit
- Repeat
  - Set next input bit for DAC to 1
  - Wait for DAC and comparator to stabilize
  - If the DAC output (test voltage) is **smaller** than the input then set the current bit to 1, else clear the current bit to 0



ARM

ADC - Successive Approximation Conversion …

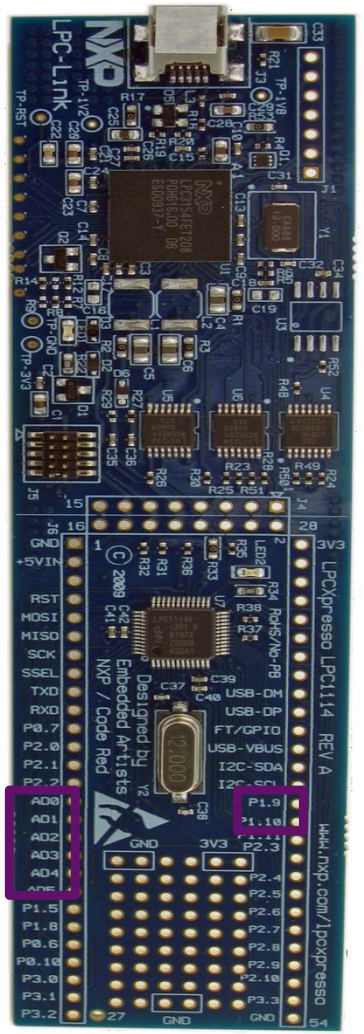




ARM

LPCxpresso 1115 Analog I/O

- Single 10-bit successive approximation ADC with 8 input channels
- AD0-AD5,AD6(PIO1_10),AD7(PIO1_11), multiplexed pins
- No on board DAC
- Measure range from 0 to 3.6V
- Power-down mode
- Individual result registers for each A/D channel to reduce interrupt overhead



ARM

Conversion Resolution and Time

- Conversion resolution is defined by both reference voltage and CLKS bits
  - By default, CLKS is 0x0 and the reference voltage is 3.3 V
  - Which means the result will be 10 bits
  - $3.3V/2^{10}=3.2mV$

- Conversion time is defined by both CLKDIV bits and CLKS bits
  - By default, CLKDIV is 0x0 and CLKS is 0x0 (11 clock cycles)
  - Which means the clock for ADC is 4.5MHz/1 (CLKDIV+1)
  - 11×1/4.5MHz=2.44 μs (shortest time)

Using the ADC

- ADC initialization
  - Enable clock
  - Enable ADC(power)
  - Configure I/O
  - Select trigger source
  - Select input channel
  - Select other parameters
- Trigger conversion
- Read results
- Calibrate? Average?

On-off Control

- For power efficiency, the ADC module is usually turned off (even if it is clocked).
- Good practical to shut down ADC whenever you are not using it.
- Two related registers: SYSAHBCLKCTRL register and PDRUNCFG register

- The first step is to disable the power down bit to the ADC block in PDRUNCFG register:
  - LPC_SYSCON->PDRUNCFG &= ~(0x1<<4);
- Then enable the clock, set the bit 13 in the System AHB clock control register since ADC clocked by APB clock (PCLK 4.5MHz) ,
  - LPC_SYSCON->SYSAHBCLKCTRL |= (1<<13);

Configuration IOCON register for ADC

- As mentioned previously, I/O needs proper configuration for different function.
- AD0 for example:

IOCON_R_PIO0_11

Table 85. IOCON_R_PIO0_11 register (IOCON_R_PIO0_11, address 0x4004 4074) bit description

Bit	Symbol	Value	Description	Reset value
2:0	FUNC		Selects pin function. All other values are reserved.	000
		0x0	Selects function R. This function is reserved. Select one of the alternate functions below.	
		0x1	Selects function PIO0_11.	
		0x2	Selects function AD0.	
		0x3	Selects function CT32B0_MAT3.	

- Choose function AD0(0x2)
  - LPC_IOCON->R_PIO0_11 &= ~0x8F; // Clear corresponding IOCON
  - LPC_IOCON->R_PIO0_11 |= 0x0A; // Analog input and pull-down mode

ARM

ADC Registers

- A/D Control Register (AD0CR),
- A/D Global Data Register (AD0GDR), most recent conversion result
- A/D Interrupt Enable Register (AD0INTEN), generate interrupt if needed
- A/D Channel x Data Register (AD0DRx, x could be 0-7)
- A/D Status Register (AD0STAT)

Interrupt Caused By ADC

- Data needed to be read by the end of conversion. Continuously fetch from the data register is a less effective way.
- Interrupt at completion of conversion
- A/D Interrupt Enable register
  - 7:0 ADINTEN, Set 1 to x bit to generate interrupt when x channel is converted.
  - 8 ADGINTEN, Set 1 to generate interrupt when any channel is converted. (Reset value 1, must be set to 0 in burst mode)
- Do whatever is needed in the handler