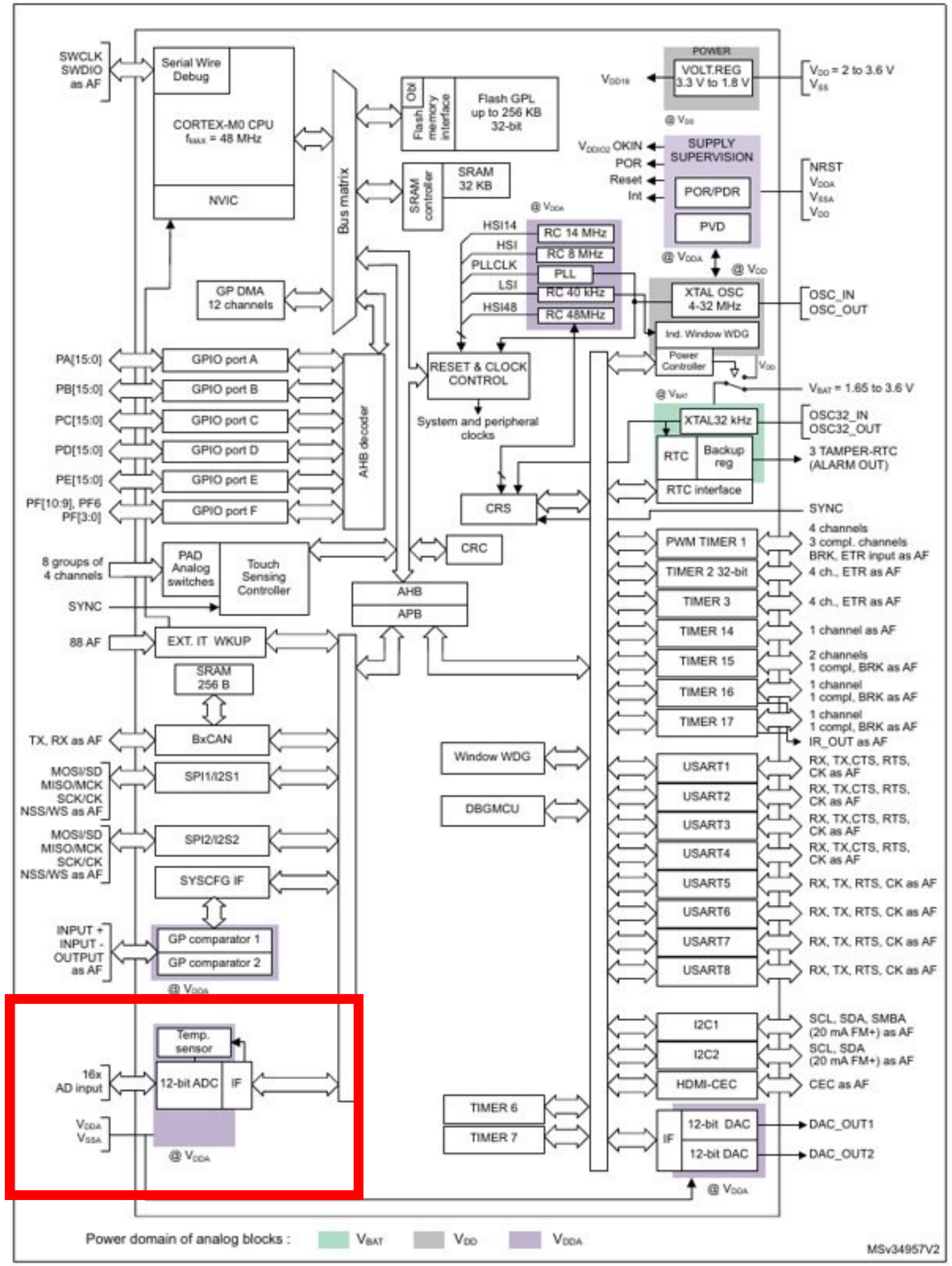
Module 3

Analog to Digital Conversion

Tim Rogers 2022

Reading Assignment

- Analog-to-Digital Conversion:
  - Text, Chapter 20, Analog-to-digital conversion. pp 481 – 506
  - Family Reference Manual, Chapter 13, "Analog-to-digital converter (ADC)", pp. 229 – 268
    - Skim it. Understand the control registers.
  - Family Reference Manual, Appendix A.7
    - Skim it. Examples of how to use the ADC.
- Coming up: Pulse-Width Modulation:
  - Textbook, Chapter 15, “General-purpose Timers”, pages 373 – 414.
  - Family Reference Manual, Chapter 17, "General purpose timers (TIM2 and TIM3)", pages 377 – 443.

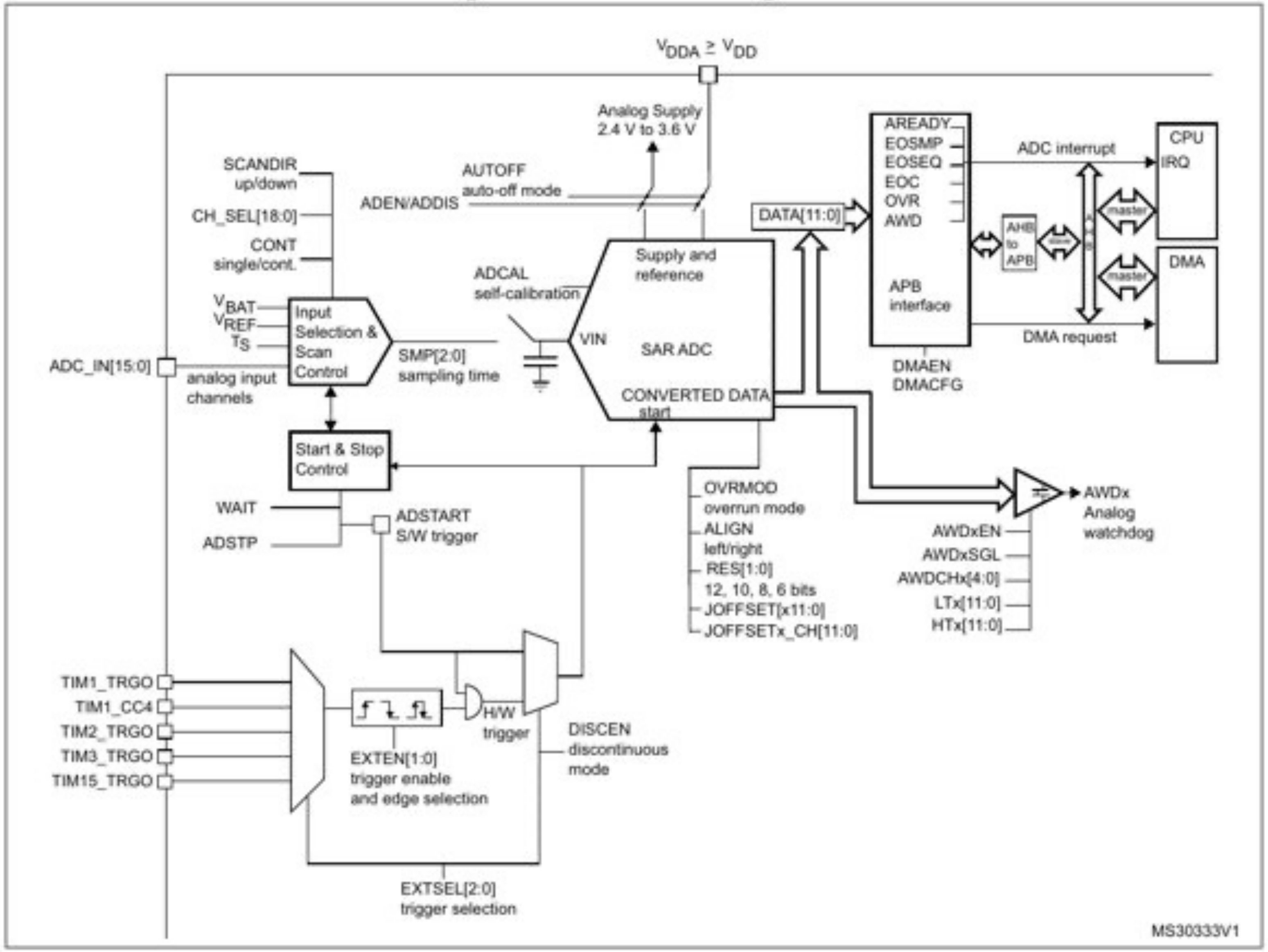


The STM32

- Analog to Digital Converter
  - 16 external channels
  - 3 internal channels
    - temperature sensor
    - reference voltage
    - power supply pin monitor
  - 0 – 3.6V conversion range
    - Our voltage reference is 3.0V
  - 12-bit resolution, by default
  - 1MHz sampling frequency

ADC Block Diagram

Figure 26. ADC block diagram



List of ADC_IN pins

- ADC_IN0: PA0
- ADC_IN1: PA1
- ADC_IN2: PA2
- ADC_IN3: PA3
- ADC_IN4: PA4
- ADC_IN5: PA5
- ADC_IN6: PA6
- ADC_IN7: PA7
- …

- ADC_IN8: PB0
- ADC_IN9: PB1
- ADC_IN10: PC0
- ADC_IN11: PC1
- ADC_IN12: PC2
- ADC_IN13: PC3
- ADC_IN14: PC4
- ADC_IN15: PC5

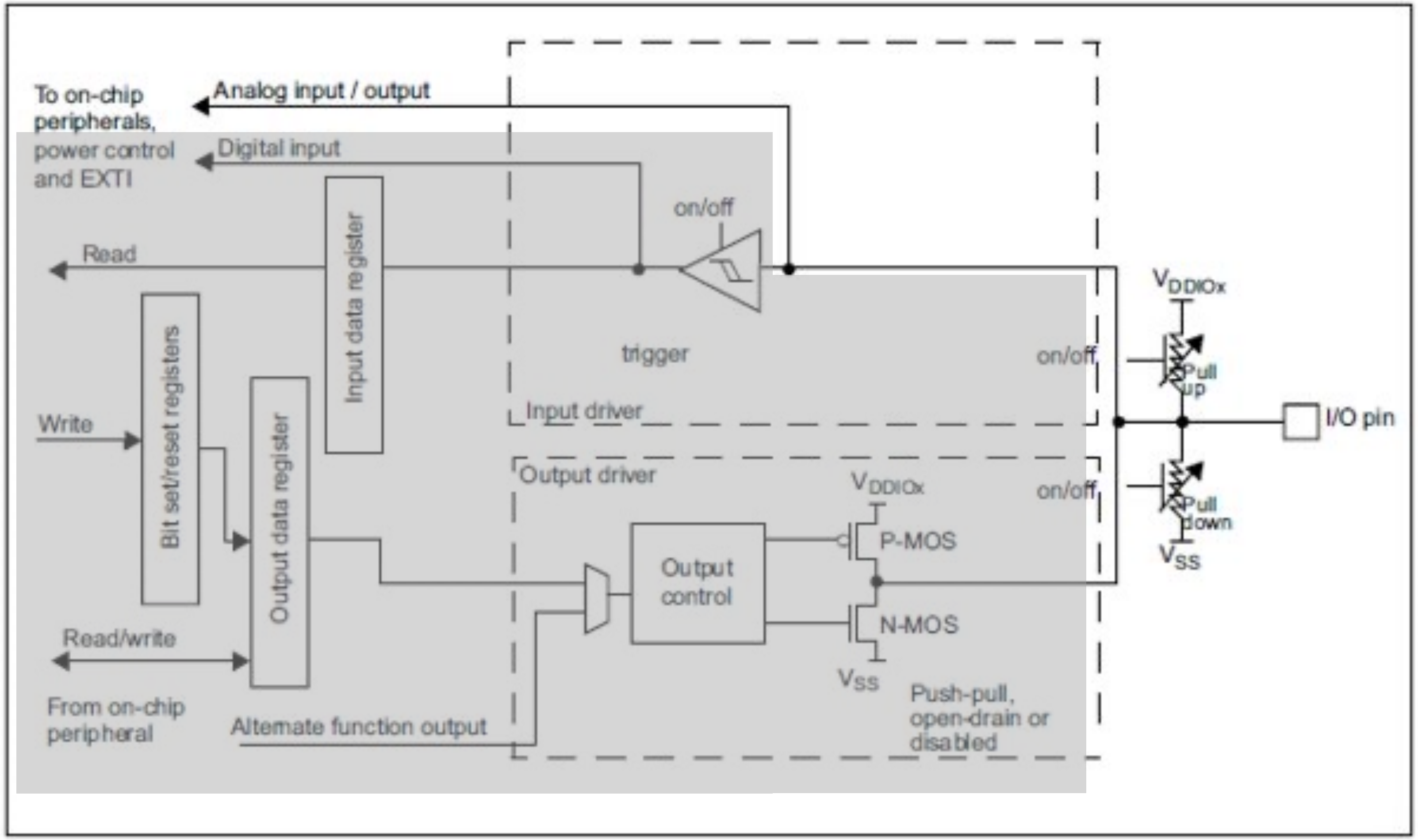
Port MODER

- 16 2-bit values determine, input, output, or special operation.
  - 00: Port pin is used for input
  - 01: Port pin is used for output
  - 10: Port pin has an alternate function
  - **11: Port pin is an analog pin**

Offset	Register																
0x00	**GPIOx_MODER** (where x = **B..F**)	MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]	MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]
	Reset value	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0	0 0

Analog path through GPIO pin

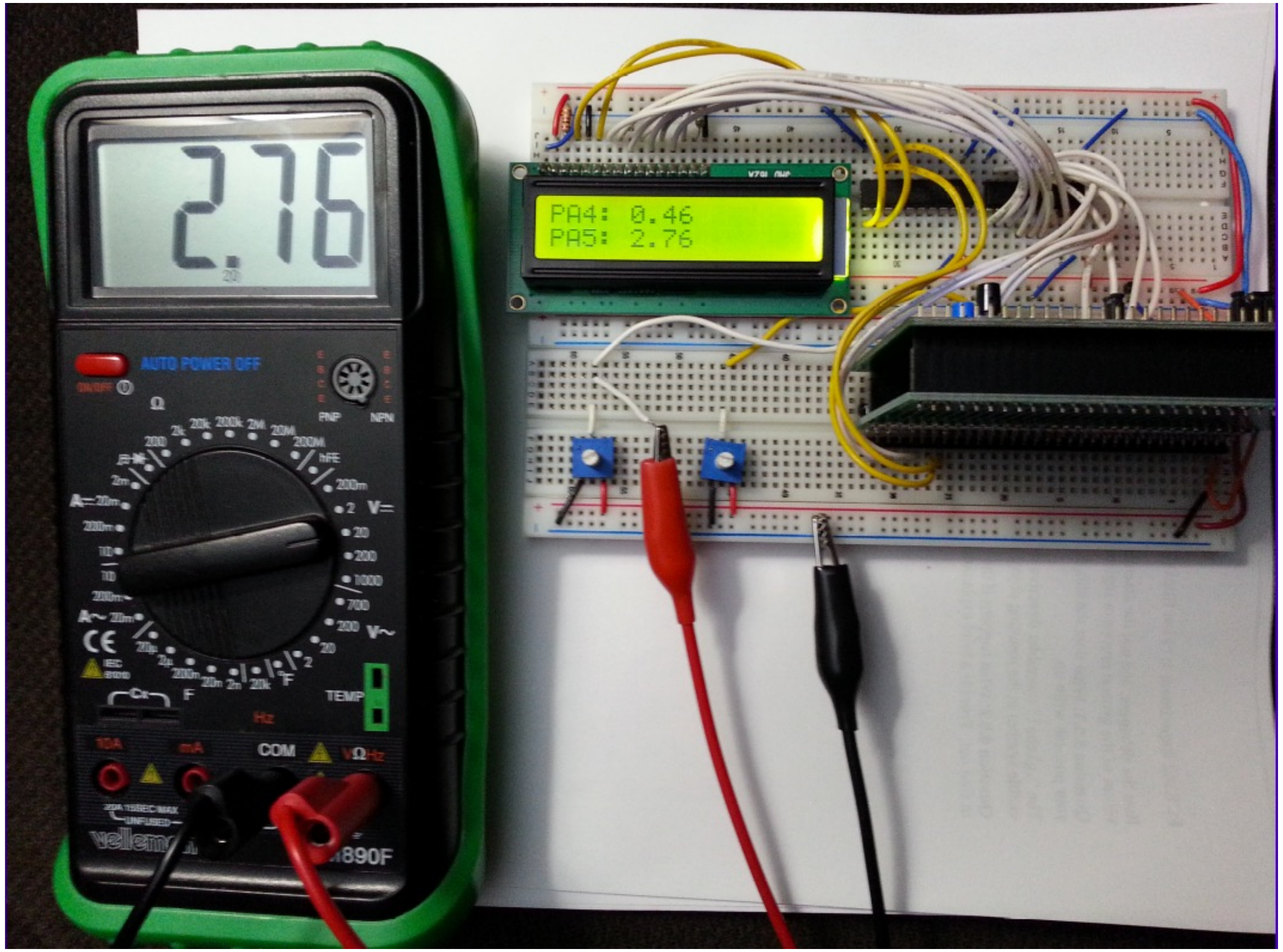
- Using a pin as an analog input



Protect the analog circuitry of the microcontroller!

- **CAUTION**: When you configure a pin on the STM32 for analog operation, you connect that pin to sensitive electronics inside the chip.
- If that pin is connected to voltage higher than 4 V for just an instant, it will permanently, irreversibly damage either the pin or the entire microcontroller.

Demo



- Two potentiometers connected between Vdd and Vss.
  - DMM also connected to PA5.
- Continuously sampled (alternating).

Things noticed in demo

- The displayed values for analog readings changed frequently. The least significant digit changes so fast, it's not readable. (Called "jitter")
- When showing raw 12-bit readings, the numbers vary from 0 to about 4025.
  - The maximum value should be 4095 (212-1) but it never reaches that value.

ADC demo

- When we have team projects, many teams make video games (which is fine).
  - For input, they might use an analog joystick.
    - Two potentiometers

Enabling an ADC pin

- Update RCC_AHBENR to turn on the clock to the appropriate GPIO port.
- Set pin type in GPIOx_MODER to ‘11’
- Enable the clock to the ADC unit.
  - Set RCC_APB2ENR’s ADC1EN bit to ‘1’.
- Enable the 14MHz high-speed internal clock.
  - Set RCC_CR2’s HSI14ON to ‘1’.
  - Wait for it to be ready by checking RCC_CR2_HSI14RDY bit.
- Activate the ADC unit.
  - Set the ADC1_CR’s ADEN bit to ‘1’
- Wait for the ADC to be “ready”.

Taking a sample

- Select a channel.
    - Set ADC1_CHSELR to zero
    - Set ADC1_CHSELR to (1 << channel#)
- Wait for the ADC to be ready.
    - Check ADC_ISR_ADRDY bit in ADC1_ISR.
- Start the conversion.
    - Turn on ADC_CR_ADSTART bit in ADC1_CR.
- Wait for the “end of conversion”.
    - Check ADC_ISR_EOC bit in ADC1_ISR.
- Read the converted value from ADC_DR.

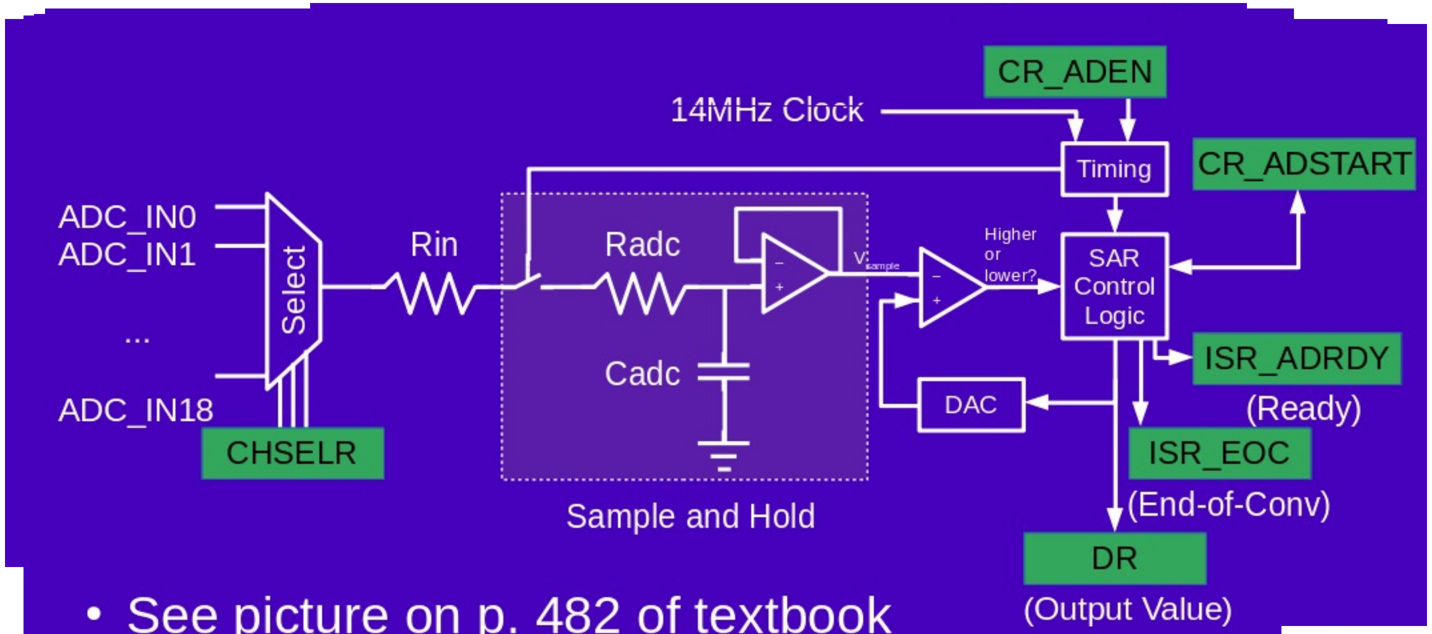
ADC Conversion takes time

- It is not "instant" like the DAC.
- The hardware uses iterative successive approximation.
  - 2 steps to “sample and hold”
  - 12 more steps to determine all 12 bits.
  - With a 14MHz clock, 1 million conversion per second.

Code for Example

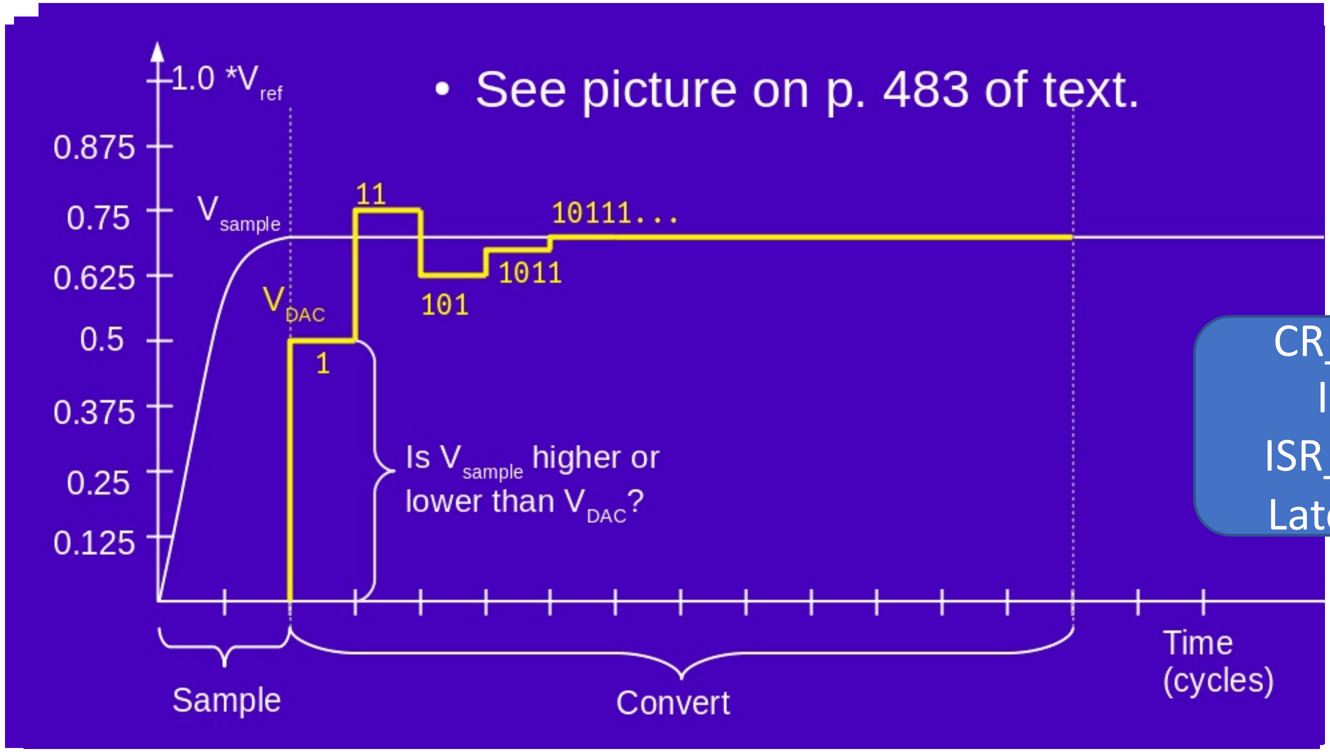
```c
int main(void)
{
    init_lcd();
    char line[21];

    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;  // Enable clock to Port A.
    GPIOA->MODER |= 0xf00;              // Set pins 5,4 for analog input.
    RCC->APB2ENR |= RCC_APB2ENR_ADC1EN; // Enable clock to ADC unit.
    RCC->CR2 |= RCC_CR2_HSI14ON;        // Turn on Hi-spd internal 14MHz clock.
    while(!(RCC->CR2 & RCC_CR2_HSI14RDY));// Wait for 14MHz clock to be ready.
    ADC1->CR |= ADC_CR_ADEN;            // Enable ADC.
    while(!(ADC1->ISR & ADC_ISR_ADRDY));// Wait for ADC to be ready.
    while((ADC1->CR & ADC_CR_ADSTART)); // Wait for ADCstart to be 0.
    while(1) {
        ADC1->CHSELR = 0;               // Unselect all ADC channels.
        ADC1->CHSELR |= 1 << 4;         // Select channel 4.
        while(!(ADC1->ISR & ADC_ISR_ADRDY));    // Wait for ADC ready.
        ADC1->CR |= ADC_CR_ADSTART;             // Start the ADC.
        while(!(ADC1->ISR & ADC_ISR_EOC));      // Wait for end of conversion.
        sprintf(line, "PA4: %2.2f", ADC1->DR * 3 / 4095.0);
        println(line);
        ADC1->CHSELR = 0;               // Unselect all ADC channels.
        ADC1->CHSELR |= 1 << 5;         // Select channel 5.
        while(!(ADC1->ISR & ADC_ISR_ADRDY));    // Wait for ADC ready.
        ADC1->CR |= ADC_CR_ADSTART;             // Start the ADC.
        while(!(ADC1->ISR & ADC_ISR_EOC));      // Wait for end of conversion.
        sprintf(line, "PA5: %2.2f", ADC1->DR * 3 / 4095.0);
        println(line);
    }
}
```

How does an ADC work?



- See picture on p. 482 of textbook

Gradual Convergence of SAR ADC

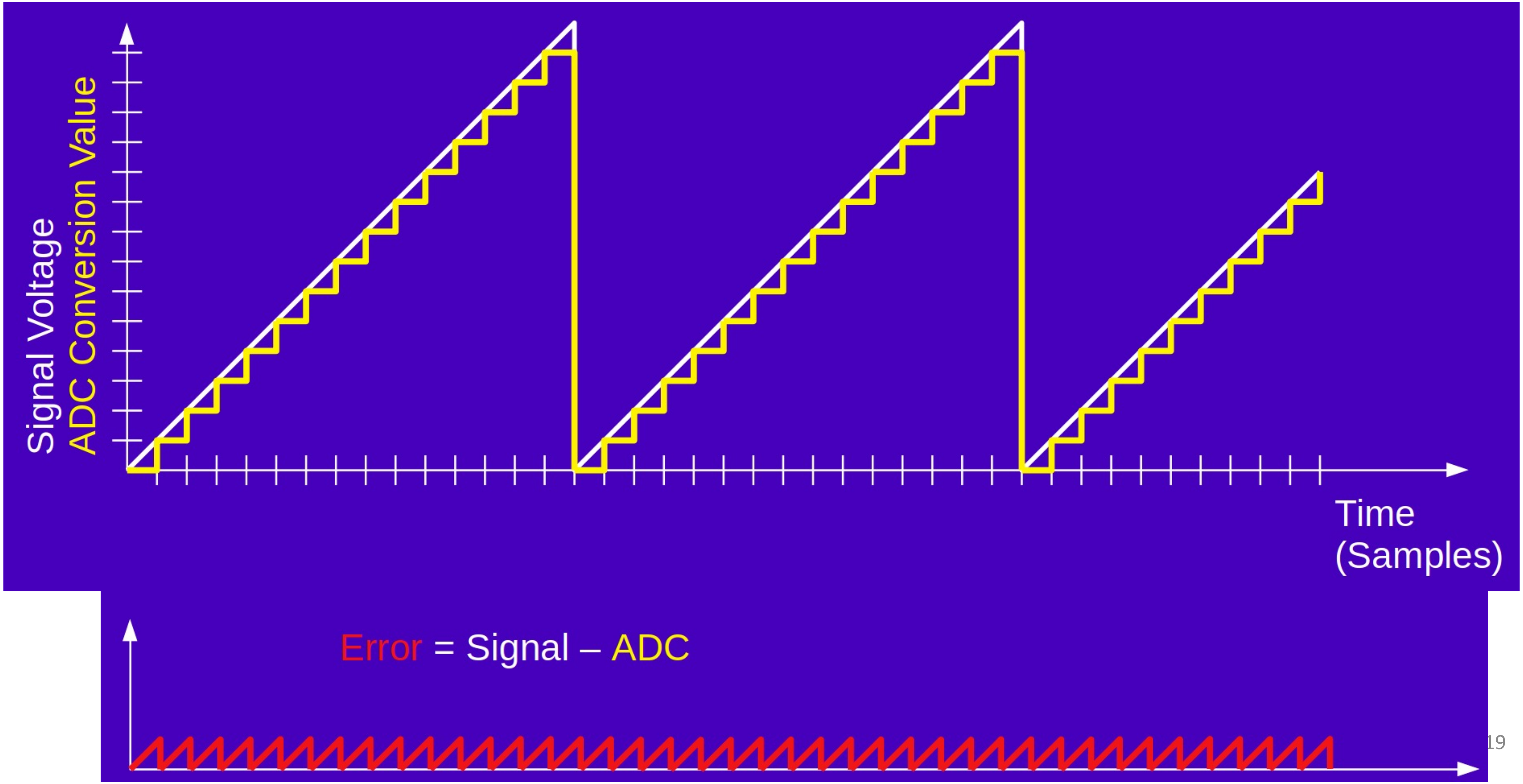


CR_ADSTART = 0
ISR_EOC = 1
ISR_ADREADY = 1
Latch value in DR

ADC Sampling Error

- ADC output is very specific, but not precise.
  - Analog signals have a continuum of values.
  - Any analog value between two digital values represents some margin of error.
  - What does this error look like?

Error over progressive samples



Out-of-range values

- An ADC value of 4095 represents VREF and a value of 0 represents VSS. What about input values higher or lower than these limits?
  - The ADC’s internal DAC cannot produce voltages less than 0 or greater than VREF, so input values are clipped at the high end or low end.

Absolute Maximum Ratings

Table 13. STM32F091xB/xC pin definitions (continued)

Pin numbers						Pin name (function upon reset)	Pin type	I/O structure	Notes	Pin functions	
UFBGA100	LQFP100	UFBGA64	LQFP64	WLCSP64	LQFP48/UFQFPN48					Alternate functions	Additional functions
L4	31	G4	22	G5	16	PA6	I/O	TTa		SPI1_MISO, I2S1_MCK, TIM3_CH1, TIM1_BKIN, TIM16_CH1, COMP1_OUT, TSC_G2_IO3, EVENTOUT, USART3_CTS	ADC_IN6

Table 12. Legend/abbreviations used in the pinout table

Name	Abbreviation	Definition
Pin name	Unless otherwise specified in brackets below the pin name, the pin function during and after reset is the same as the actual pin name	
Pin type	S	Supply pin
	I/O	Input / output pin
I/O structure	FT	5 V-tolerant I/O
	FTf	5 V-tolerant I/O, FM+ capable
	TTa	3.3 V-tolerant I/O directly connected to ADC
	TC	Standard 3.3 V I/O
	RST	Bidirectional reset pin with embedded weak pull-up resistor

Absolute maximum ratings

Stresses above the absolute maximum ratings listed in *Table 21: Voltage characteristics*, *Table 22: Current characteristics* and *Table 23: Thermal characteristics* may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these conditions is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

Table 21. Voltage characteristics(1)

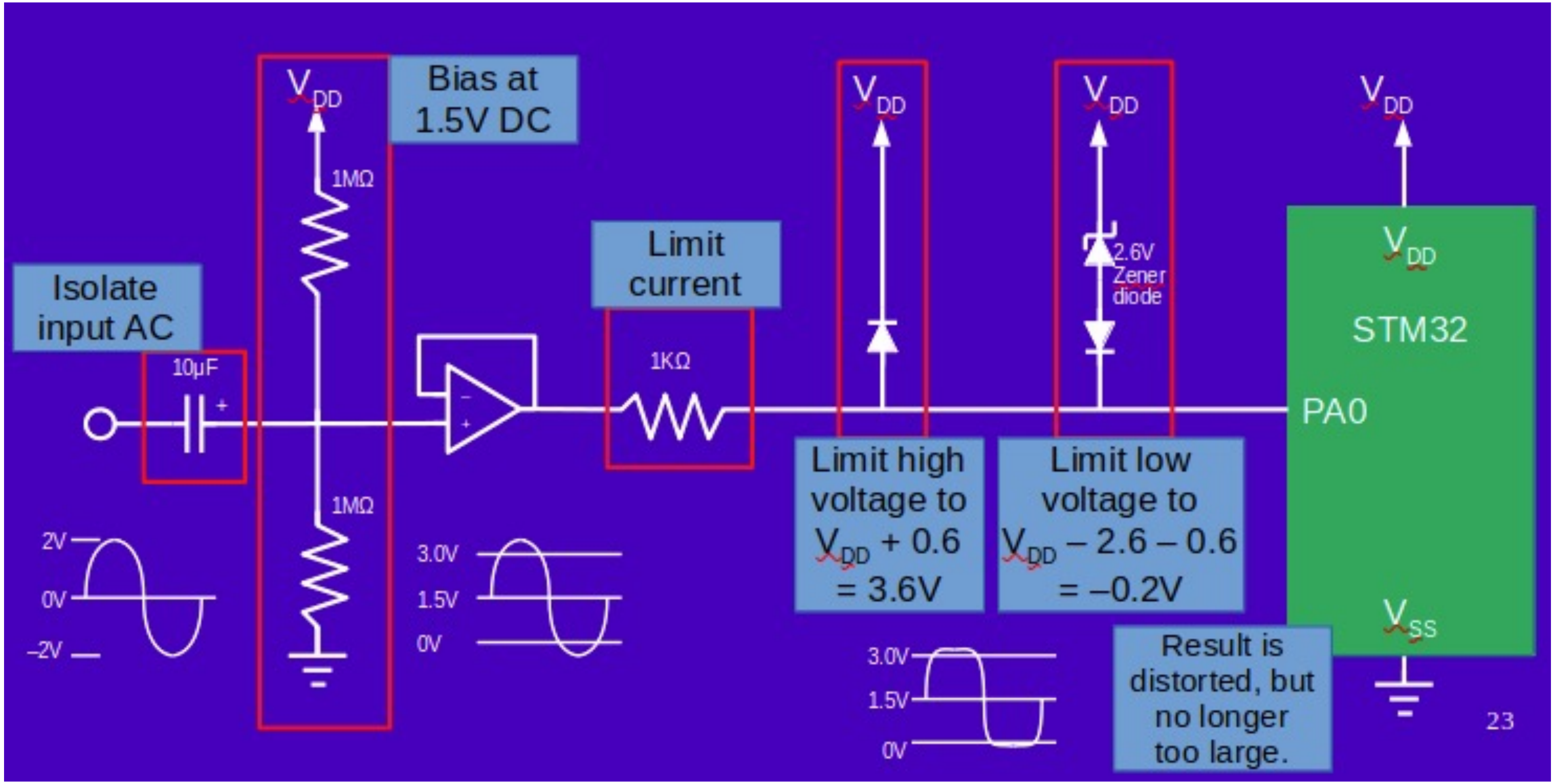
Symbol	Ratings	Min	Max	Unit
$V_{DD}-V_{SS}$	External main supply voltage	- 0.3	4.0	V
$V_{DDIO2}-V_{SS}$	External I/O supply voltage	- 0.3	4.0	V
$V_{DDA}-V_{SS}$	External analog supply voltage	- 0.3	4.0	V
$V_{DD}-V_{DDA}$	Allowed voltage difference for $V_{DD} > V_{DDA}$	-	0.4	V
$V_{BAT}-V_{SS}$	External backup supply voltage	- 0.3	4.0	V
V_{IN}(2)	Input voltage on FT and FTf pins	V_{SS} - 0.3	V_{DDIOx} + 4.0 (3)	V
	Input voltage on TTa pins	V_{SS} - 0.3	4.0	V
	Input voltage on any other pin	V_{SS} - 0.3	4.0	V
$\lvert\Delta V_{DDx}\rvert$	Variations between different V_{DD} power pins	-	50	mV
$\lvert V_{SSx} - V_{SS}\rvert$	Variations between all the different ground pins	-	50	mV
$V_{ESD(HBM)}$	Electrostatic discharge voltage (human body model)	see *Section 6.3.12: Electrical sensitivity characteristics*		-

1. All main power (V_{DD}, V_{DDA}) and ground (V_{SS}, V_{SSA}) pins must always be connected to the external power supply, in the permitted range.
2. V_{IN} maximum must always be respected. Refer to for the maximum allowed injected current values.
3. Valid only if the internal pull-up/pull-down resistors are disabled. If internal pull-up or pull-down resistor is enabled, the maximum limit is 4 V.

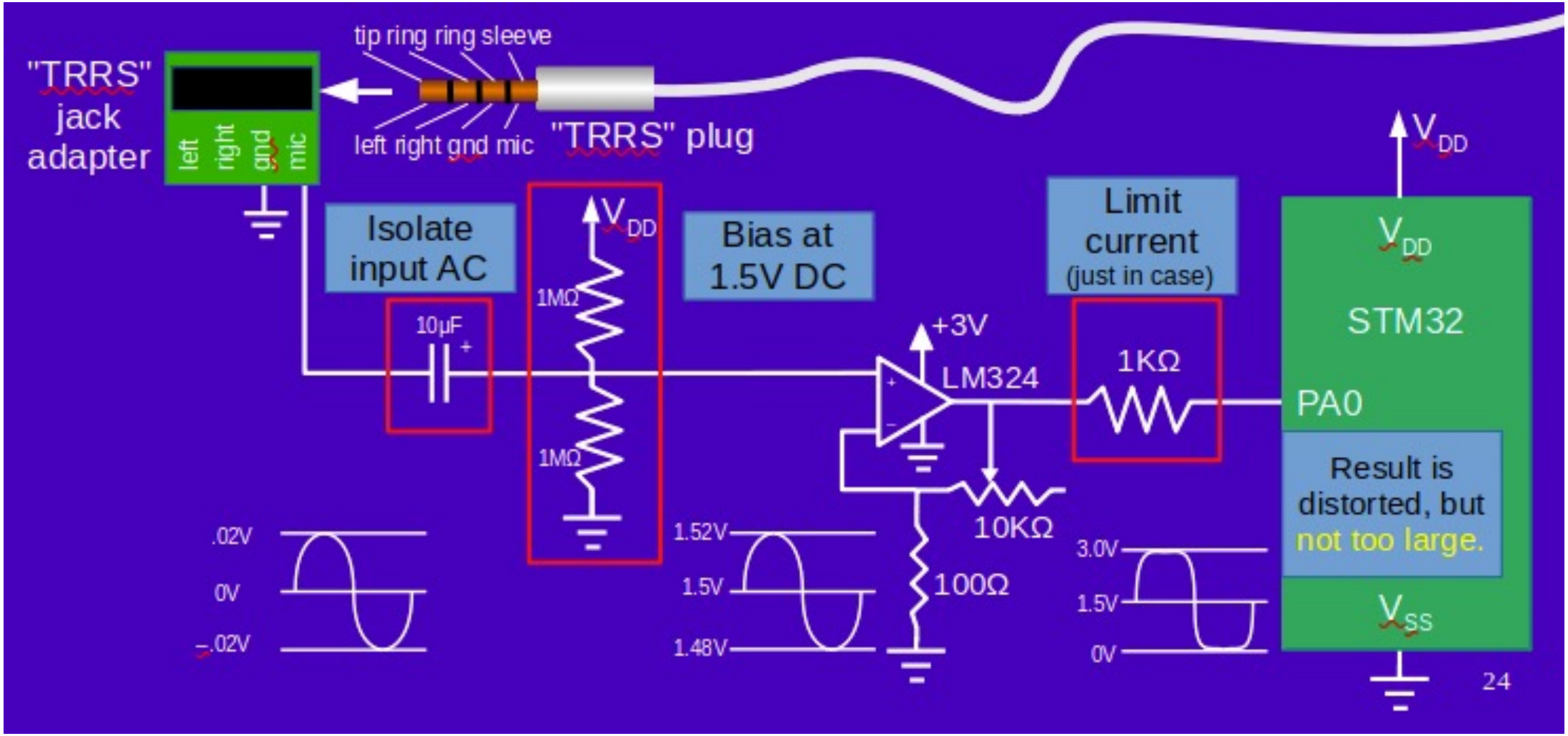
Absolute Maximum Ratings

- **Remember**: Going past the Absolute Maximum Ratings may mean you will **permanently, irreparably damage** a pin or the entire microcontroller

How can we limit input to ADC?



Microphone Input For ADC



Chosing a Sampling Rate

- In ECE 301 (or 440), you will learn about the Nyquist Rate.
  - You must sample a signal twice as fast as the maximum frequency you want to represent.
- Real world sample rate examples:
  - Compact Disc encoding: 44.1kHz, (16-bit per sample, per channel)
    - Fills 32K of memory in 0.2 seconds
  - Telephone network: 8kHz, (8-bit)
    - Fills 32K of memory in 4 seconds

Left or Right Alignment

- Samples placed in the ADC_DR can be left-aligned or right-aligned with 6-, 8-, 10-, or 12-bit resolution.

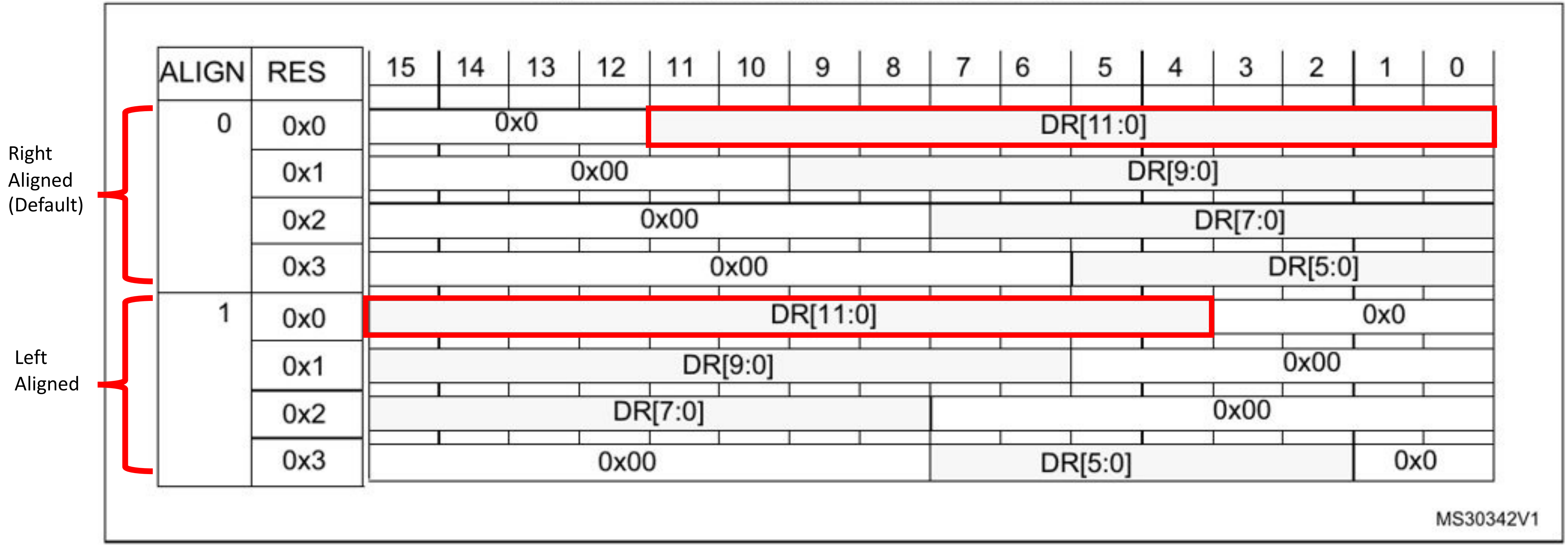


Figure 37. Data alignment and resolution

"Jitter"

jitter

n 1: small rapid variations in a waveform
resulting from fluctuations in the voltage
supply or mechanical vibrations or other
sources

2: a small irregular movement

Why the jitter in the ADC result?

- I used to believe it was because of bad power supply design:
- The voltage reference for the STM32 chip (VDDA) is derived from a Schottky diode drop.
- The same line that powers the chip.
- The more current consumed by the STM32, the larger the voltage drop.
- If the VDDA changes, a constant ADC_INx voltage will appear to shift.

Jitter is always there

- Jitter is noise from the surroundings, from the microcontroller, and from the ADC itself.

Defeating Jitter

- One of the most reliable ways of defeating noise is taking more samples than needed and averaging them out
  - e.g., for a 1 kHz update rate, take 128k samples/sec, and average the last 128 samples
  - Called "boxcar averaging"

Efficient Boxcar Averaging

```c
#define HISTSIZE 128
    int hist1[HISTSIZE] = { 0 };
    int sum1 = 0;
    int pos1 = 0;
    int hist2[HISTSIZE] = { 0 };
    int sum2 = 0;
    int pos2 = 0;
...
    reading = ADC1->DR;
    sum1 -= hist1[pos1];
    sum1 += hist1[pos1] = reading;
    pos1 = (pos1 + 1) % HISTSIZE;
    val = sum1/HISTSIZE;
    sprintf(line, "PA4: %2.3f", val * 3 / 4095.0);
    display1(line);
```

```c
ADC1->CHSELR = 0;
ADC1->CHSELR |= 1 << 5;
while(!(ADC1->ISR & ADC_ISR_ADRDY));
ADC1->CR |= ADC_CR_ADSTART;
while(!(ADC1->ISR & ADC_ISR_EOC));


reading = ADC1->DR;
sum2 -= hist2[pos2];
sum2 += hist2[pos2] = reading;
// if HISTSIZE is a power of 2, replace divisions with
// a bitwise AND operation and right shift
pos2 = (pos2 + 1) & (HISTSIZE-1);
val = sum2 >> 7;
sprintf(line, "PA5: %2.3f", val * 3 / 4095.0);
display2(line);
```

Pragmatic Use of ADC

- The canonical examples of how to use the ADC involve lots of waiting for ready, starting, waiting, checking for end-of-conversion, etc.
  - About the same thing for the DAC as well.
- As long as you know that your sample rate is less than the conversion rate of the ADC, **<u>don't bother</u>** waiting or checking.
  - Similar strategy used with the DAC in the polyphonic sound generation notes.

Example Pragmatic ISR

```c
// Assume ADC initialization is proper and complete.

void ISR(void) {
    int x = ADC1->DR;»  »   »   // Read the completed ADC value.
    ADC1->CR |= ADC_CR_ADSTART;»// Start the next conversion.
    // Do something with the value read...
    // Do other work in the ISR...
}

// As long as ISR is not invoked more than 1M times per second,
// the value will be ready on the next invocation of the ISR.

// This is more complicated when sampling from more than one input.
```

More Efficient Transfers

- The largest overhead with respect to ADC transfer as described is the latency of invocation of, and return from, the ISR. (Same with the DAC.)
  - Due to saving and restoring registers.
- There is no way to write an ISR to do meaningful computation and read from the ADC at its maximum rate.
- Direct Memory Access (DMA) can transfer at the full rate.