ECE3411 – Fall 2016
Lecture 5a.

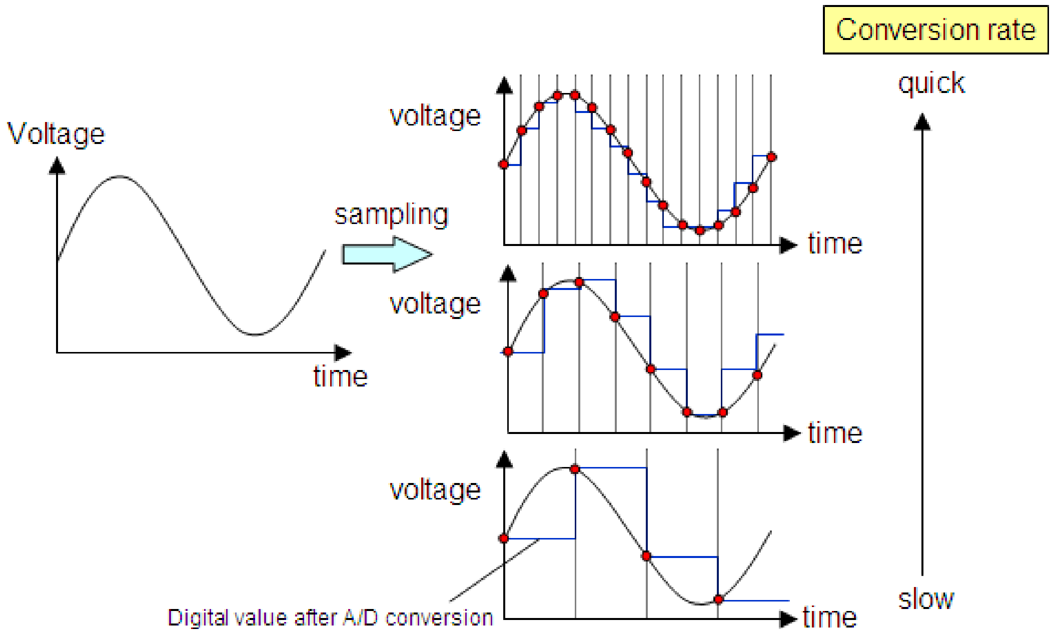
ADC: Analog to Digital Conversion

Marten van Dijk, Syed Kamran Haider
Department of Electrical & Computer Engineering
University of Connecticut
Email: {marten.van_dijk, syed.haider}@uconn.edu

Copied from Lecture 5a, ECE3411 – Fall 2015, by
Marten van Dijk and Syed Kamran Haider

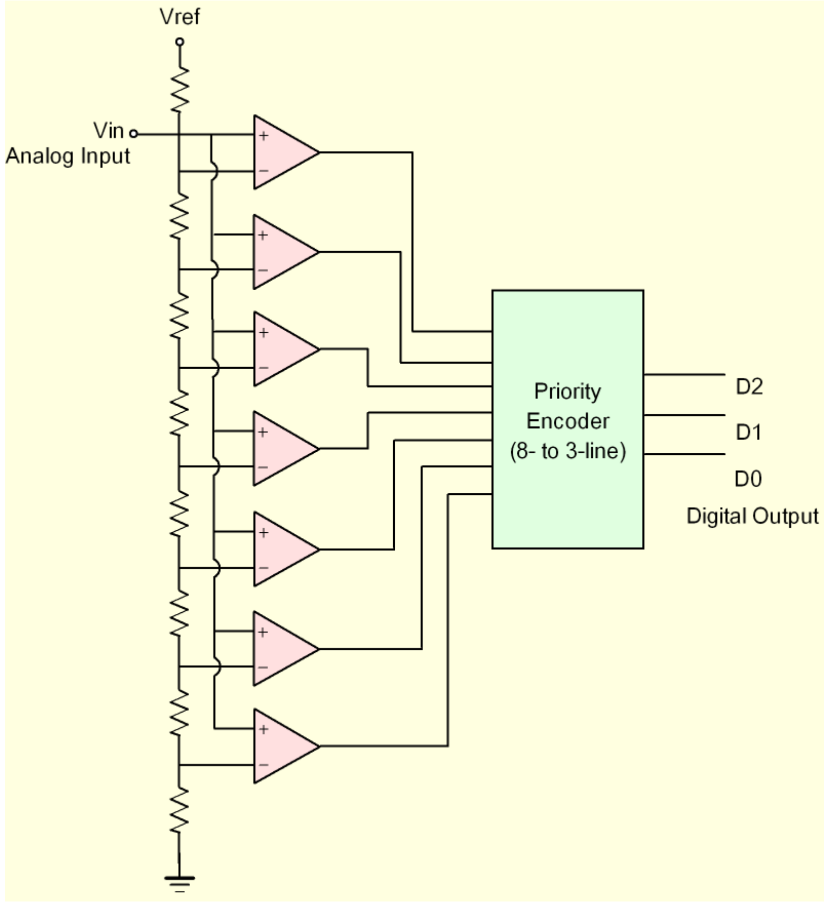
Introduction

- Why do we need Analog-Digital Conversion?
  - Real world is Analog
  - Digital computers process Digital signals
  - ADC/DAC serve as interface between Computers and Real world!
- Analog Signals are "Continuous"
  - A "Discrete" version of the analog signal is created by "Sampling" the analog signal
  - ADC then maps each sample onto a quantized range of voltages which can be represented by binary values.



ADC Types: Flash ADC

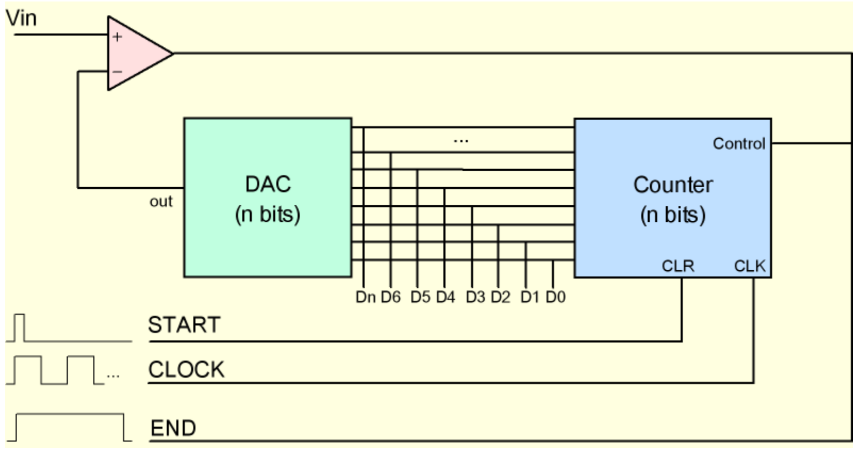
- Parallel Design
  - A resistor divider network generates discrete voltage levels
  - Input voltage is compared against all the voltage levels at once
  - Priority Encoder considers the first "HIGH" input from the top as valid, and converts it to binary form.
- Advantage: Fast
  - Conversion takes just one cycle
- Disadvantage: A lot of components needed.
  - $2^n - 1$ comparators needed for n bit ADC



Picture Source: www.hardwaresecrets.com

ADC Types: Ramp ADC

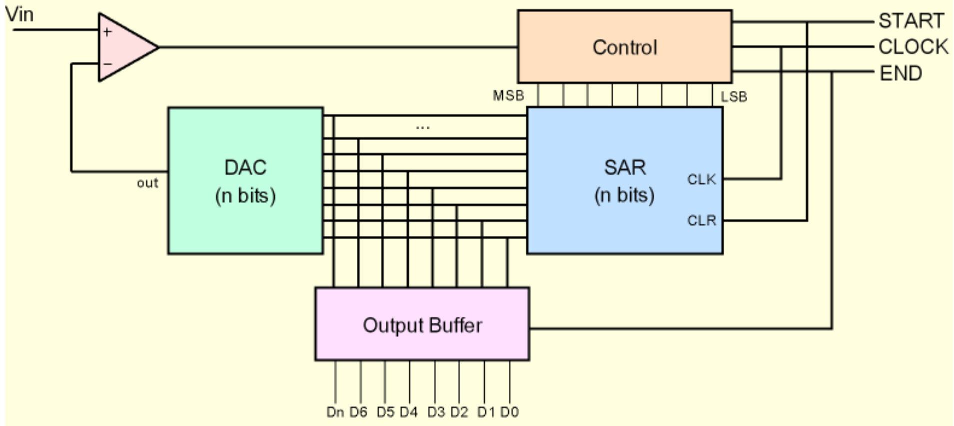
- Sequential Design
  - A Counter counts from $0 \cdots 2^n$
  - A DAC generates discrete voltage levels corresponding to the digital values $0 \cdots 2^n$ (i.e. a voltage Ramp)
  - In each cycle, input voltage is compared against the current voltage level generated by DAC
  - The comparator generates a "HIGH" value as soon as the ramp crosses the input value. The corresponding counter value becomes the output.
- Advantage: Only a few components needed.
- Disadvantage: Very slow.
  - $2^n - 1$ cycles (in worst case) for n bit ADC conversion



Picture Source: www.hardwaresecrets.com

ADC Types: Successive Approximation ADC

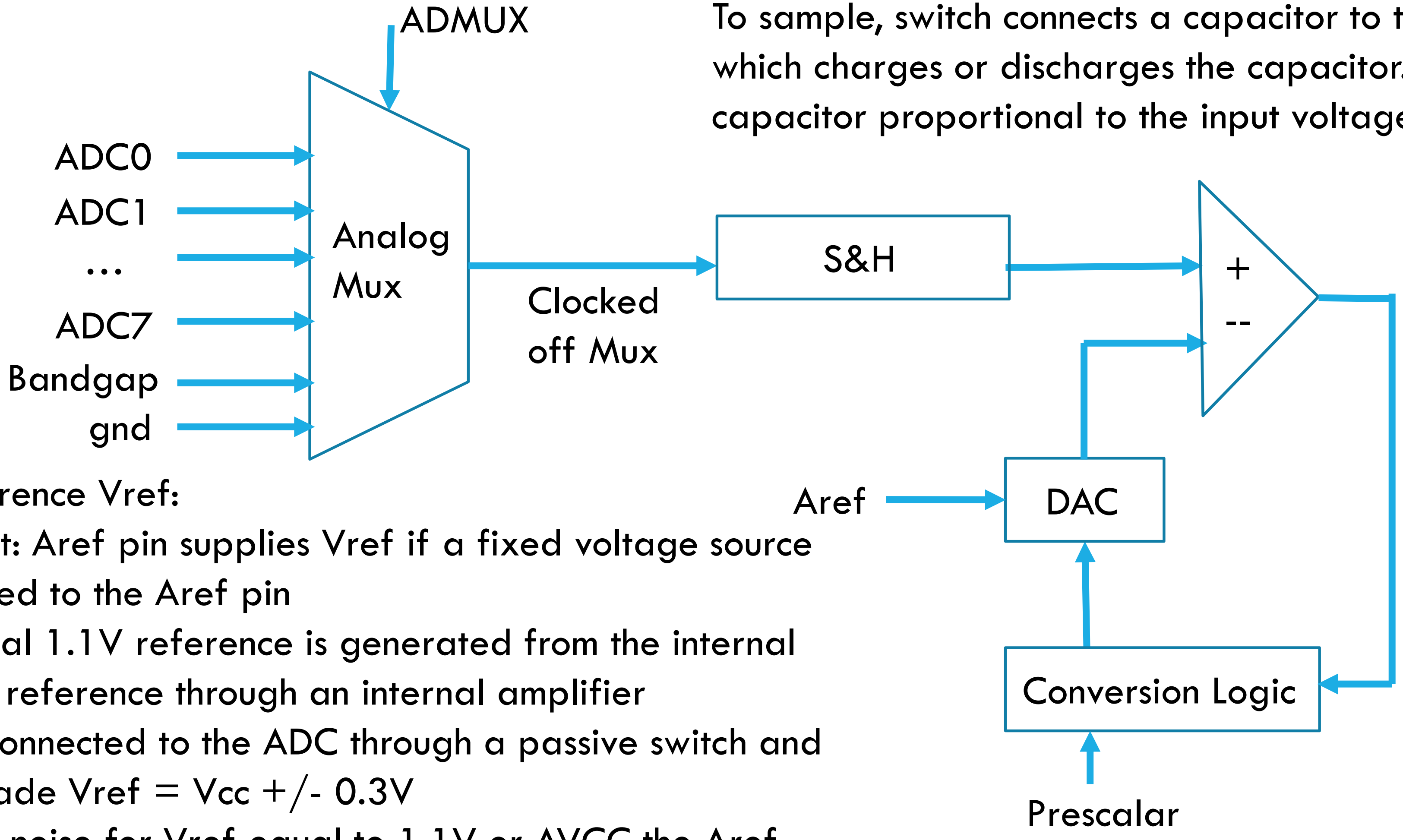
- Sequential Design
  - Closest digital value is approximated by “Binary Search”
  - First, the MSB of SAR is set to 1, and the comparator decides whether the input voltage is higher or lower than DAC voltage. The bit value is adjusted accordingly.
  - The process is repeated for each bit from MSB down to LSB
  - The final SAR value becomes the output.
- Most widely used ADC type.
- Advantages:
  - Only a few components needed.
  - Conversion takes just n cycles.



Picture Source: www.hardwaresecrets.com

ATMega328P ADC Diagram

To sample, switch connects a capacitor to the output of a buffer amplifier, which charges or discharges the capacitor. This makes voltage across the capacitor proportional to the input voltage. To hold, the switch disconnects.


ADMUX
ADC0
ADC1
…
ADC7
Bandgap
gnd
Analog Mux
Clocked off Mux
S&H
+
--
Aref
DAC
Conversion Logic
Prescalar

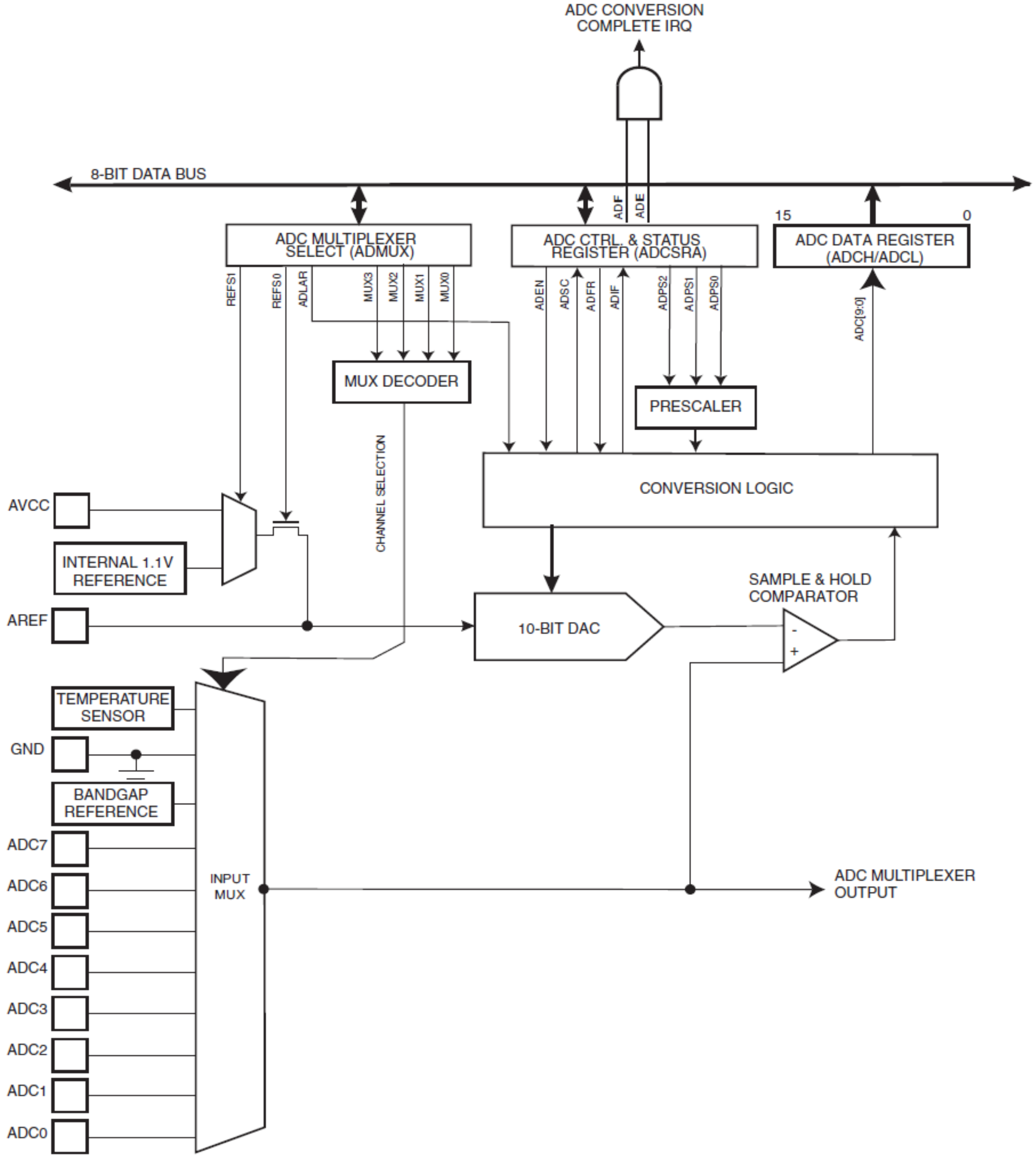
Conversion logic implements a successive approximation algorithm (a binary search; one bit per search):

- DAC takes as input the output of the conversion logic and converts it to an analog voltage where Aref sets the full range
- Analog comparator decides whether the DAC output or input voltage is the largest

Voltage reference Vref:

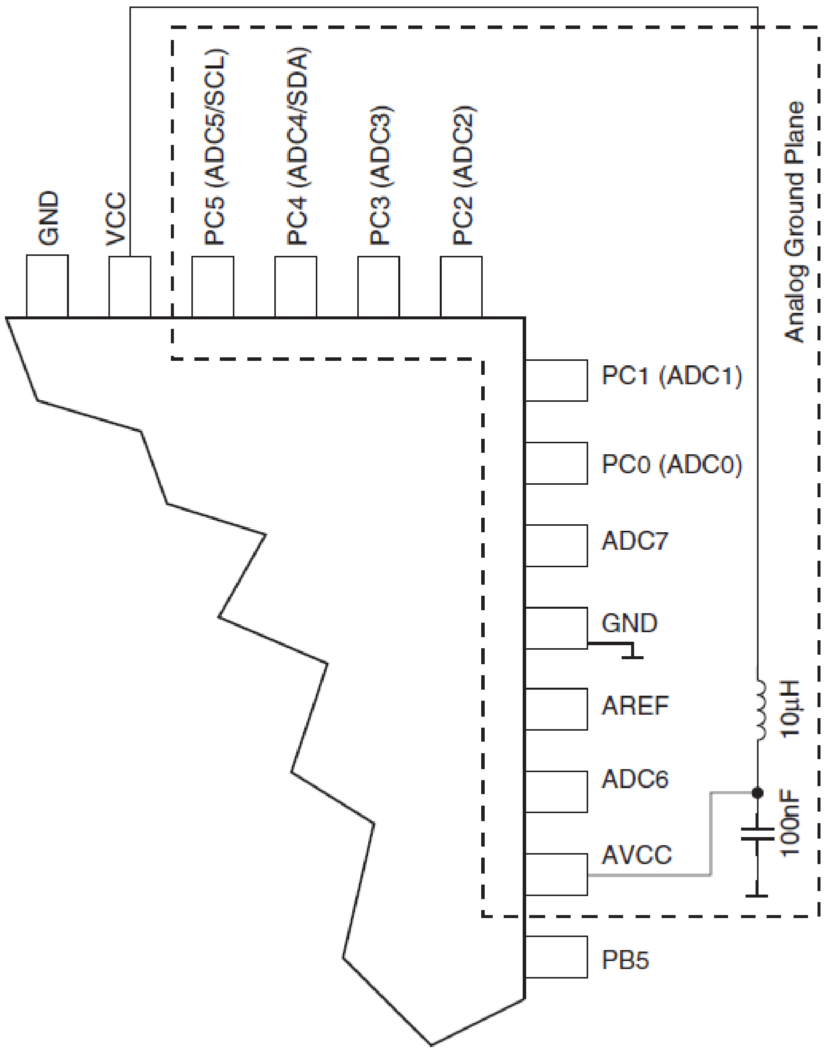
- By default: Aref pin supplies Vref if a fixed voltage source is connected to the Aref pin
- The internal 1.1V reference is generated from the internal bandgap reference through an internal amplifier
- AVCC is connected to the ADC through a passive switch and can be made Vref = Vcc +/- 0.3V
- To reduce noise for Vref equal to 1.1V or AVCC the Aref pin can be externally decoupled by a capacitor to ground

Figure 23-1. Analog to Digital Converter Block Schematic Operation,



Pin Assignment

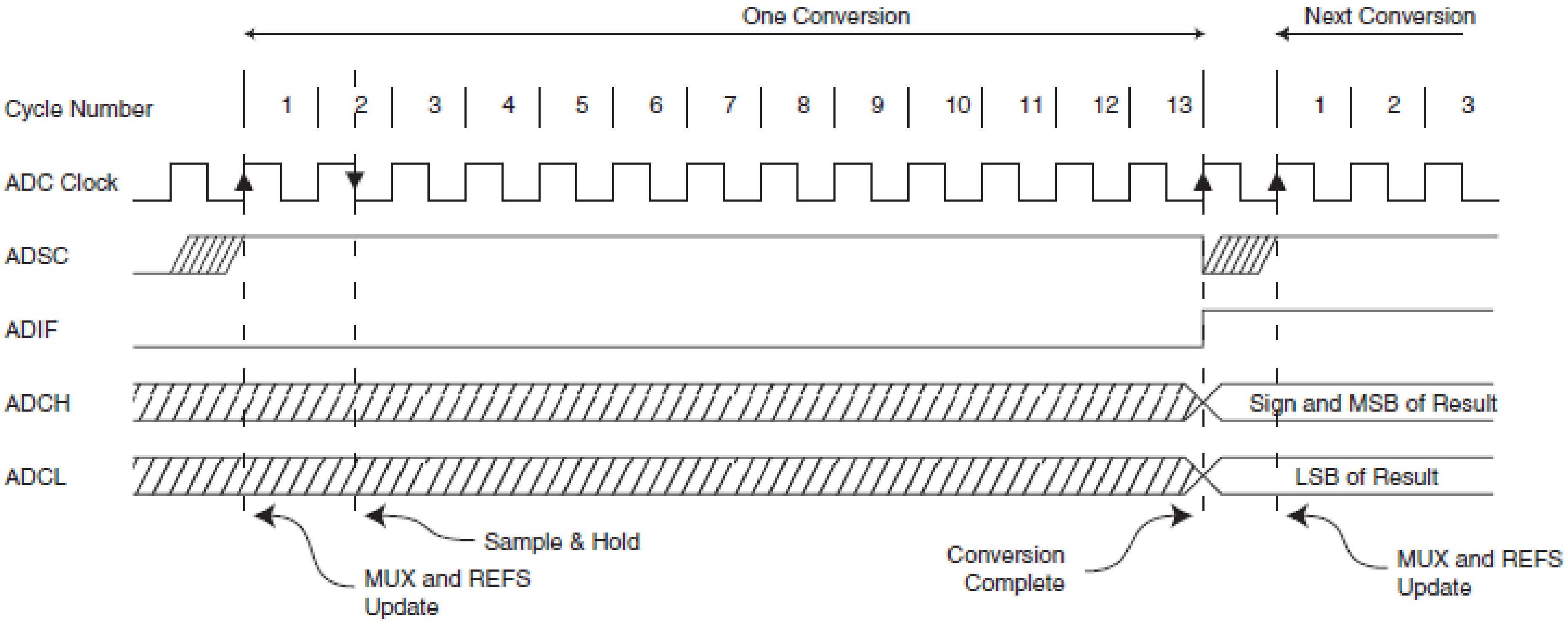
Figure 23-9. ADC Power Connections



Normal Conversion

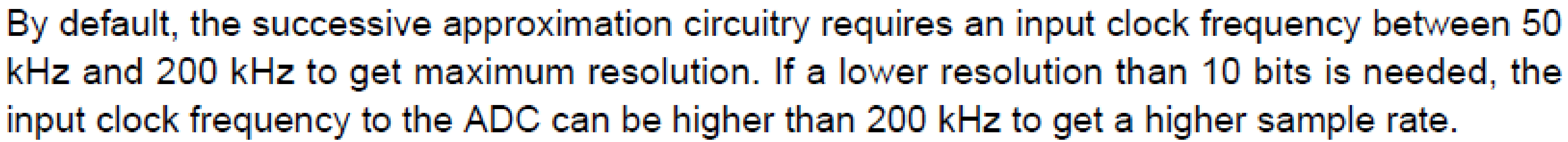
- Takes 13 cycles

Figure 23-5. ADC Timing Diagram, Single Conversion



Accuracy

- Capacitor in S&H leaks and can therefore not hold a value for too long
  - There exists a minimum sample speed/frequency
- Conversion logic takes time, so we cannot sample too fast
  - There exists a maximum sample speed/frequency
  - The faster you sample, you get a smaller number of accurate output bits (since the binary search cannot completely finish)



By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate.

- Noise: MCU produces up to 150mV line noise, there are other sources such as electrical field, etc.
  - Use capacitances close to the CPU to eliminate most of the inductance

Prescalar

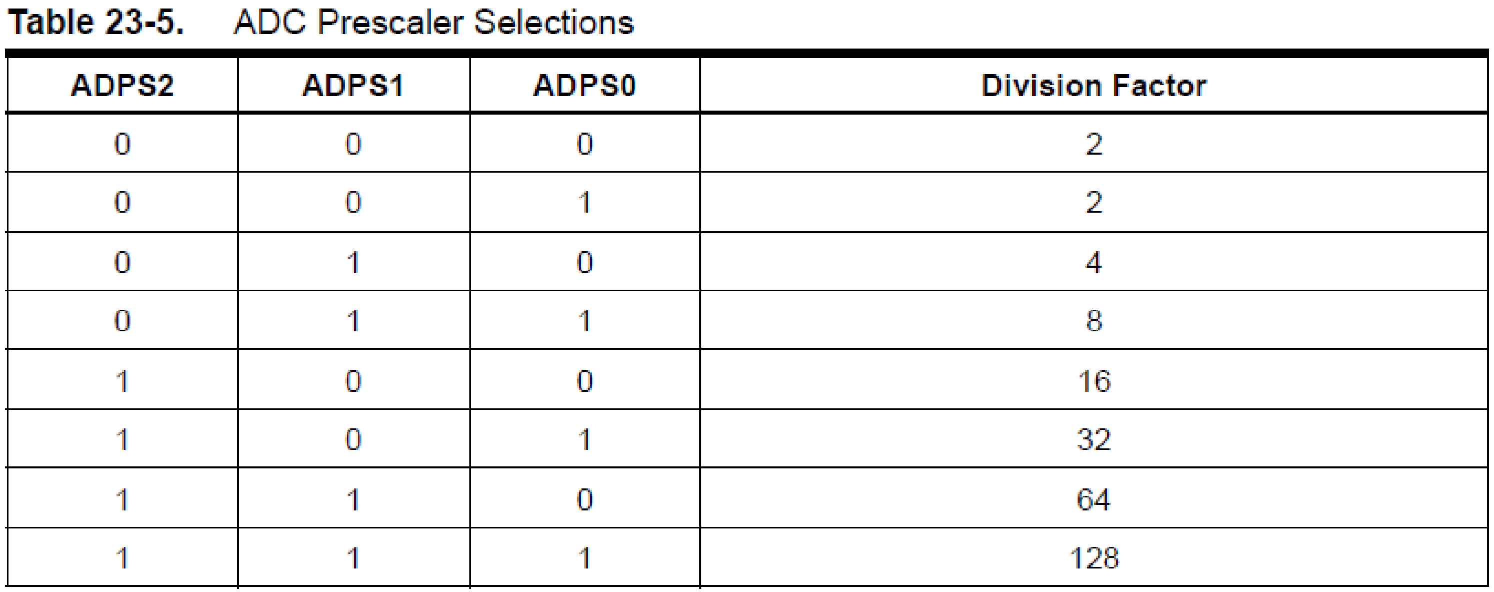


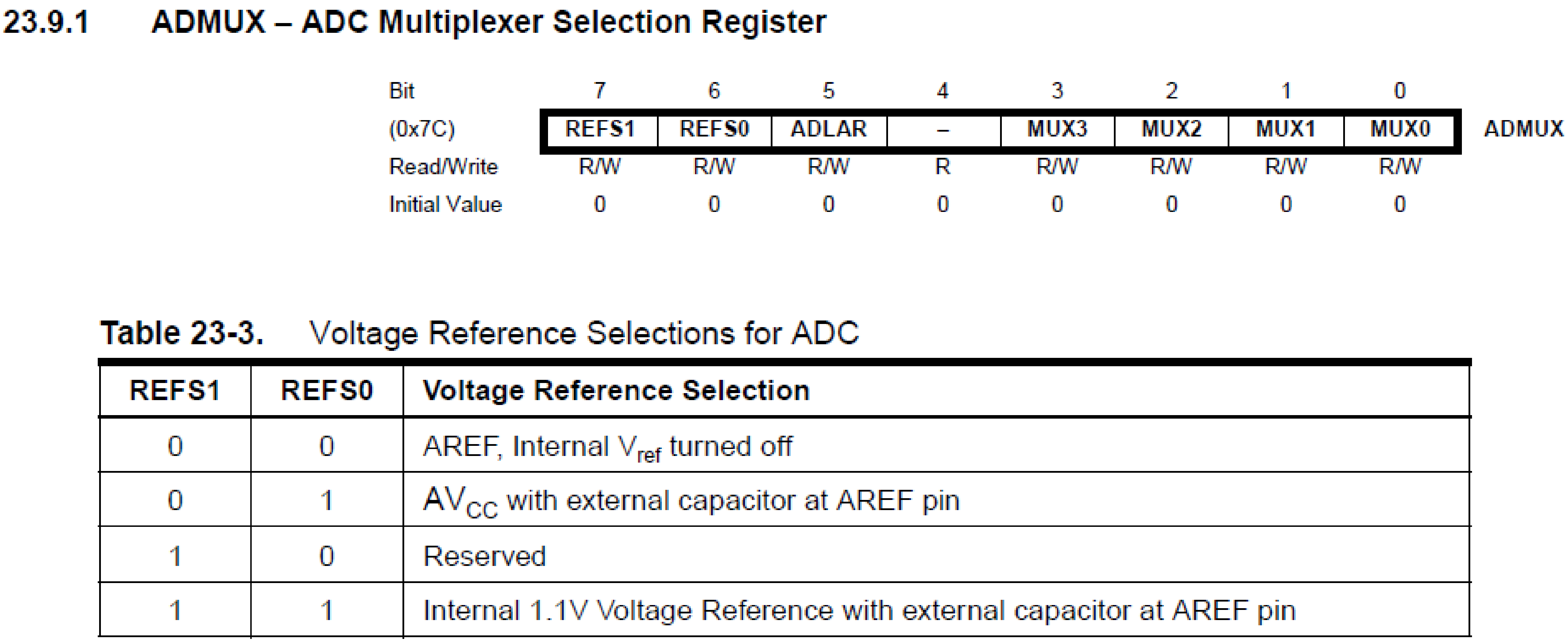
Table 23-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

- E.g., a prescalar of 128 gives 16MHz/128 = 125000 (between 50 and 200 kHz)
- To complete the binary search takes 13 cycles = 13/125000 = 104 micro seconds
- Gives 10 bits uncalibrated accuracy at a linear scale to Vref

- CPU clock is at least twice as fast as the ADC’s acceptable frequency; therefore the smallest prescalar must be >=2

ADMUX Register



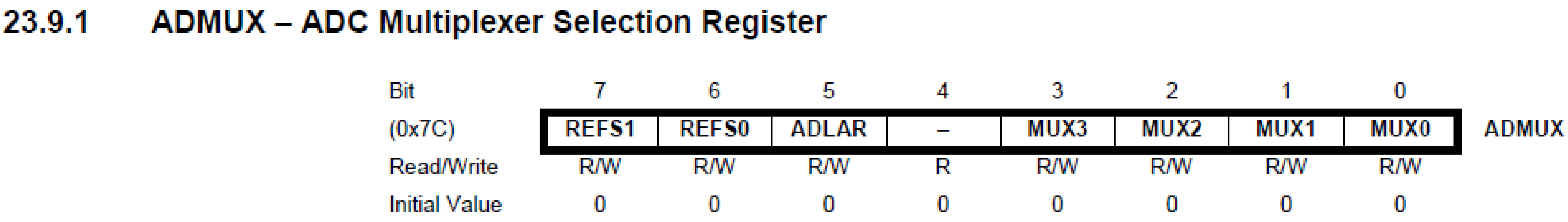
23.9.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	**REFS1**	**REFS0**	**ADLAR**	–	**MUX3**	**MUX2**	**MUX1**	**MUX0**	**ADMUX**
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 23-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

ADMUX Register



23.9.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

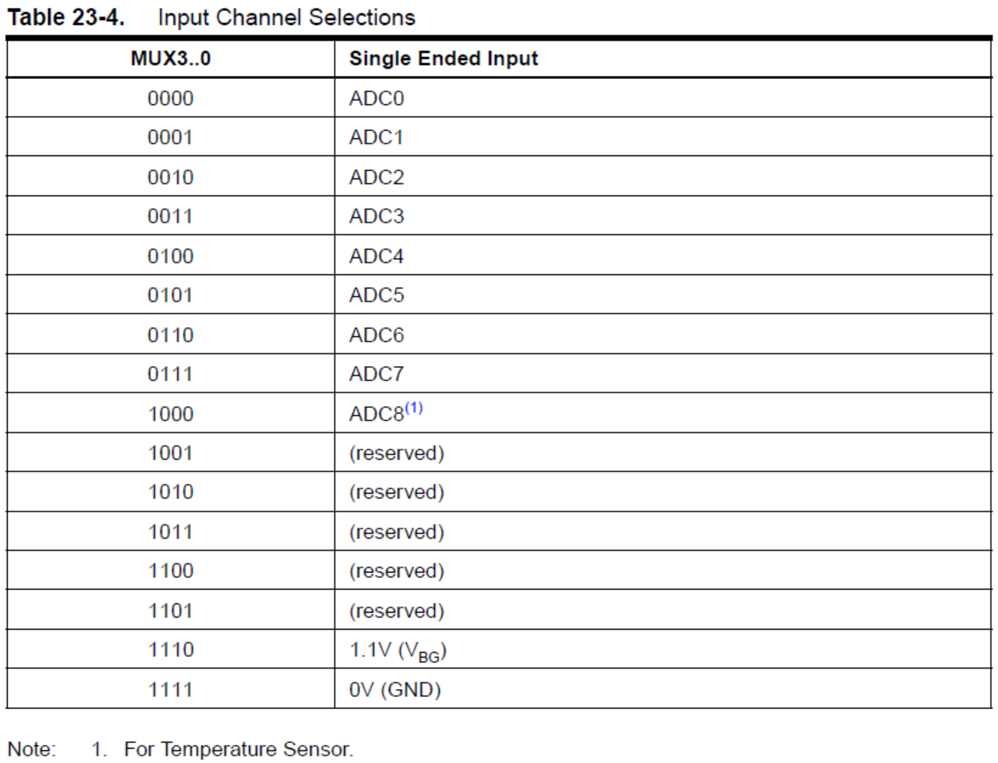


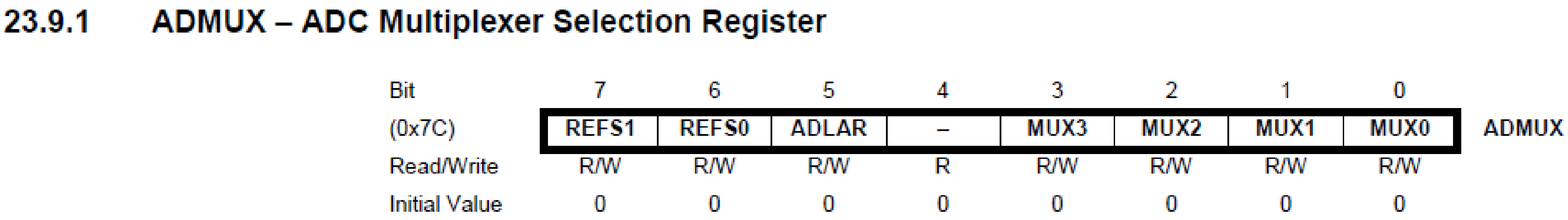
Table 23-4. Input Channel Selections

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8[1]
1001	(reserved)
1010	(reserved)
1011	(reserved)
1100	(reserved)
1101	(reserved)
1110	1.1V (V_{BG})
1111	0V (GND)

Note: 1. For Temperature Sensor.

0..7 indicate input pins ADC0 .. ADC7

ADCH/ADCL: ADC Data Registers

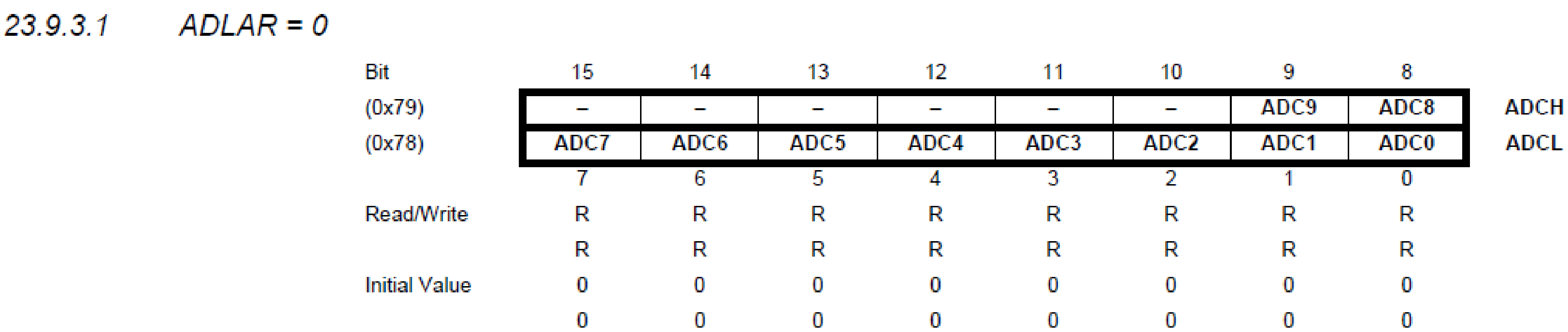


23.9.1 ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

23.9.3 ADCL and ADCH – The ADC Data Register

ADLAR = Analog Data Left Adjust Register

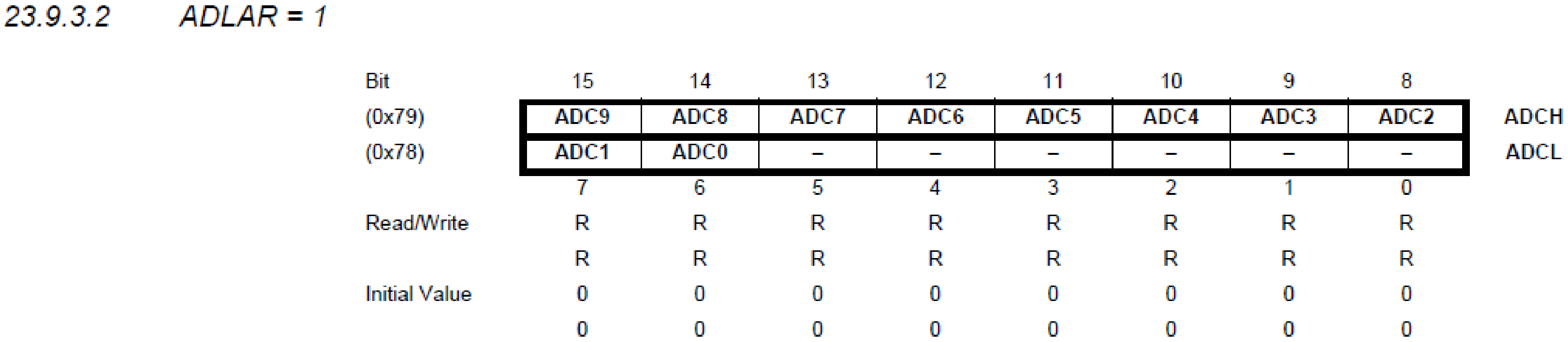


23.9.3.1 ADLAR = 0

Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

If ADLAR is set to 0,

- read ADCL for low order bits, and
- until ADCH is read the ADC is locked out

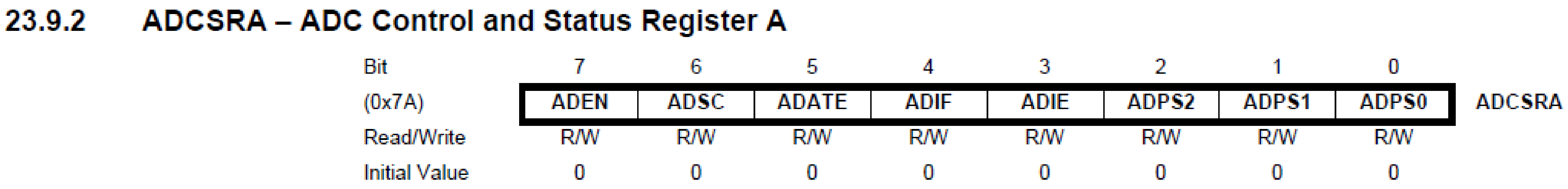


23.9.3.2 ADLAR = 1

Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

For 8-bit conversion, set ADLAR to 1 and read ADCH

ADCSRA: ADC Status Register A

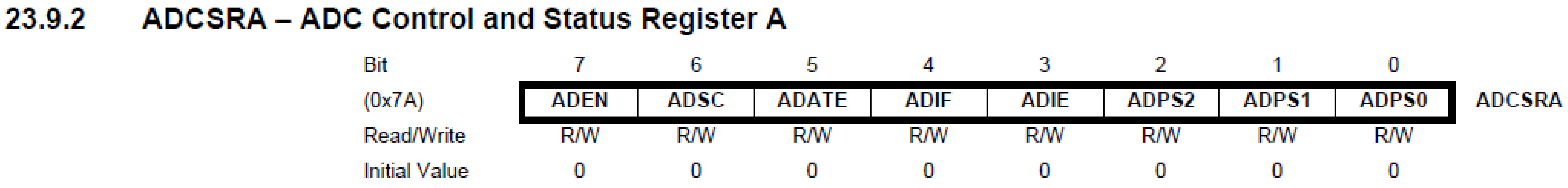


23.9.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7: ADEN – analog converter enable bit; set this bit to 1 if you want to do a conversion
- Bit 6 ADSC – AD start conversion; if it is set to 1, then a conversion is started for you and it is auto set back to 0 when done
  - You can poll this bit and as soon as it is 0, you know the conversion is done
  - Or you can poll the interrupt flag (or use the corresponding ISR if enabled):
- Bit 4: ADIF – AD interrupt flag; will be set when a conversion is done and will trigger an interrupt if ADIE is set
  - Warning: do not mess with this flag, e.g., use ADCSRA |= (1<<ADSC);

ADCSRA: ADC Status Register A

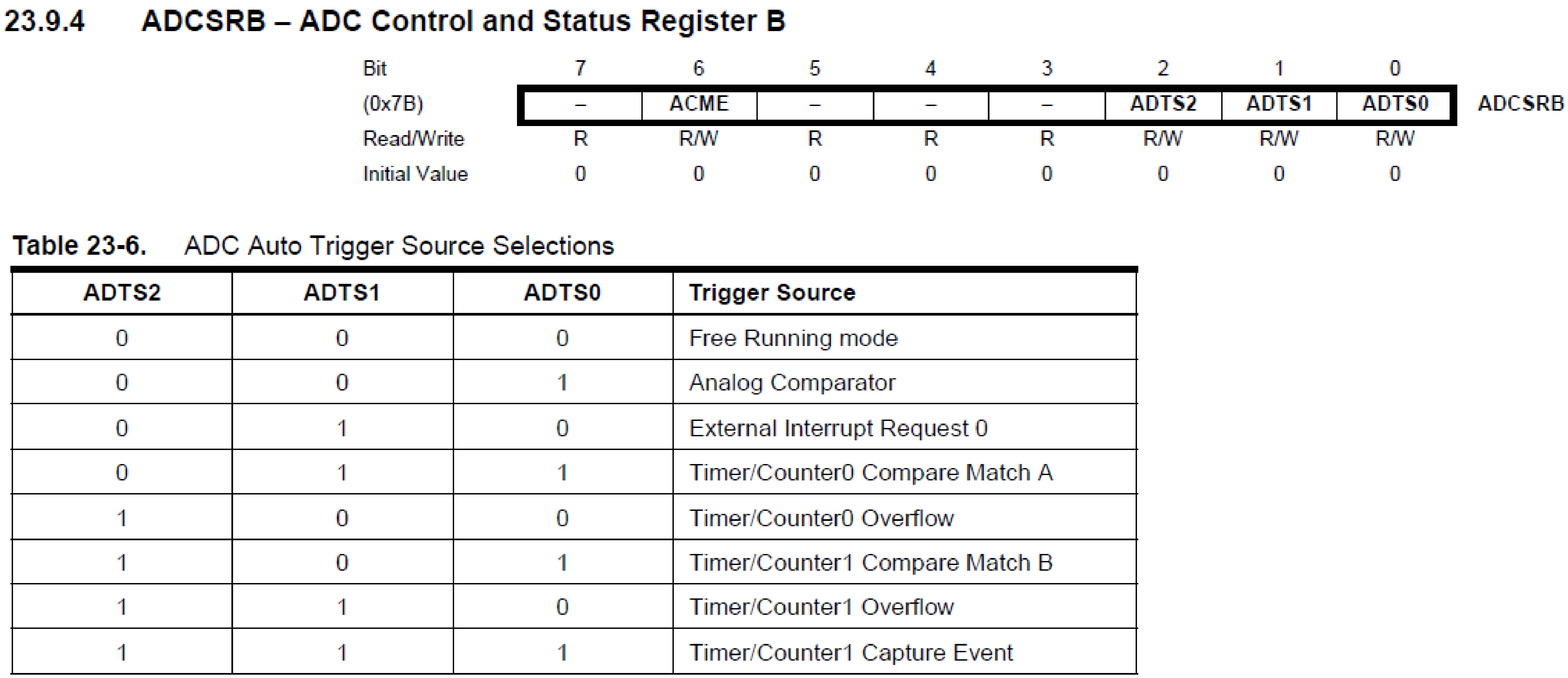


23.9.2 ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 3: ADIE – AD interrupt enable; if turned on, write the ISR to handle what happens when conversion finishes
- Bit 5: ADATE – allows one out of 8 selected events to trigger the ADC converter when coupled with the ADCSRB register
- Bits 0,1,2: prescalar (see previous slide)

ADCSRB



23.9.4 ADCSRB – ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0	
(0x7B)	–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
Read/Write	R	R/W	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 23-6. ADC Auto Trigger Source Selections

ADTS2	ADTS1	ADTS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match A
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter1 Overflow
1	1	1	Timer/Counter1 Capture Event

Example code ADC, no interrupt

```
// Borrowed from Bruce Land - Cornell University

// Performs single, left adjusted conversions and prints to UART

#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
#include <util/delay.h>
#include <math.h>
#include "uart.h"

volatile int Ain, AinLow;
volatile float Voltage;
char VoltageBuffer[6];

FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);
```

Example code ADC, no interrupt

```
void main(void)
{
        DDRC &= 0x00;    // PC1 = ADC1 is set as input

        uart_init();
        stdout = stdin = stderr = &uart_str;

        // ADLAR set to 1 → left adjusted result in ADCH
        // MUX3:0 set to 0001 → input voltage at ADC1
        ADMUX = (1<<MUX0) | (1<<ADLAR);

        // ADEN set to 1 → enables the ADC circuitry
        // ADPS2:0 set to 111 → prescalar set to 128 (104us per conversion)
        ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);

        // Start A to D conversion
        ADCSRA |= (1<<ADSC);
        fprintf(stdout,"\n\rStarting ADC demo...\n\r");
```

Takes more than 1ms, hence conversion will finish which takes 104us

Example code ADC, no interrupt

```
    while (1)
    {
        // Read from ADCH to get the 8 MSBs of the 10 bit conversion
        Ain = ADCH;

        // Typecast the volatile integer into floating type data, divide by maximum 8-bit value, and
        // multiply by 5V for normalization
        Voltage = (float)Ain/256.00 * 5.00;

        //ADSC is cleared to 0 when a conversion completes.  Set ADSC to 1 to begin a conversion.
        ADCSRA  |= (1<<ADSC);

        // Write Voltage to string format and print (3 char string + "." + 2 decimal places)
        dtostrf(Voltage, 3, 2, VoltageBuffer);
        fprintf(stdout,"%s\n\r",VoltageBuffer);
    }

    return 0;
}
```

Takes more than 1ms, hence conversion will finish which takes 104us

Conversion needs to finish

- Conversion needs to finish before the next conversion is called

- Use a print statement
- Delay functionality (of at least 104us)
- while (!(ADCSRA & (1<<ADSC) == 0)) { }
  - The most efficient solution