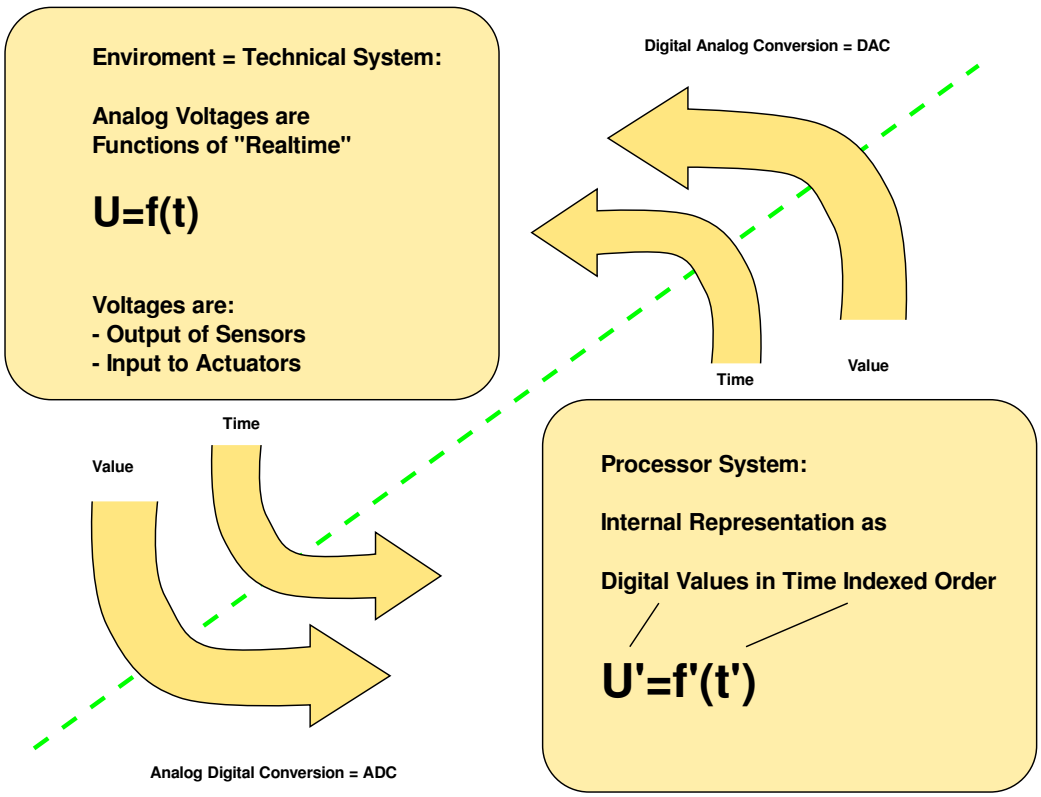
Analog Digital Converter (ADC)

Vorlesung Mikroprozessortechnik

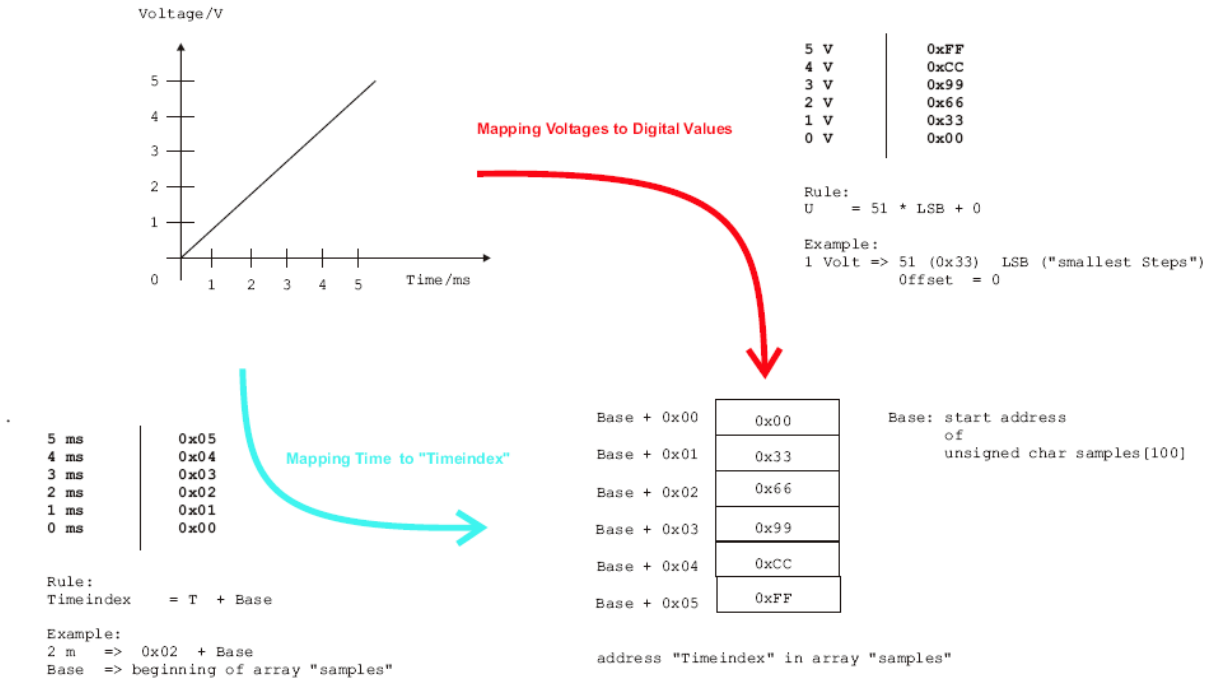
HAW Hamburg

4. Januar 2018

Analogwerte und digitale Darstellung



Analogwerte zu digitaler Repräsentation konvertieren



Digitale Repräsentation in Analogwerte konvertieren

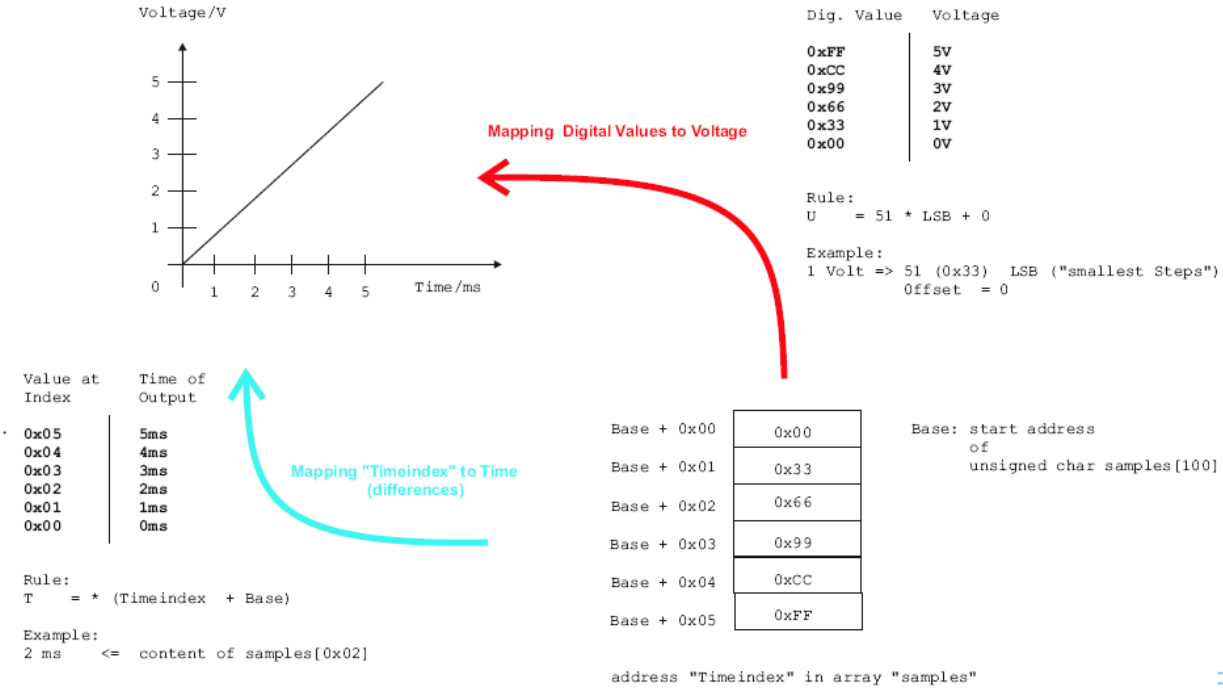


Abbildung reeller Analogwerte auf ein "Raster" von ganzzahligen Werten und diskreter Zeitpunkte

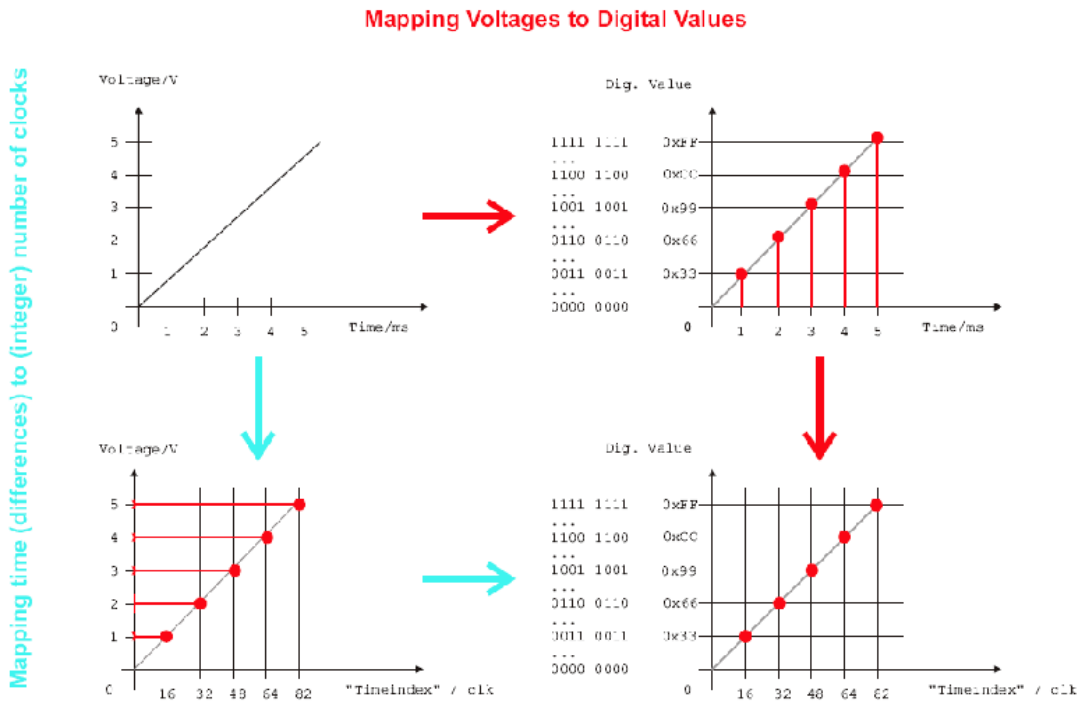
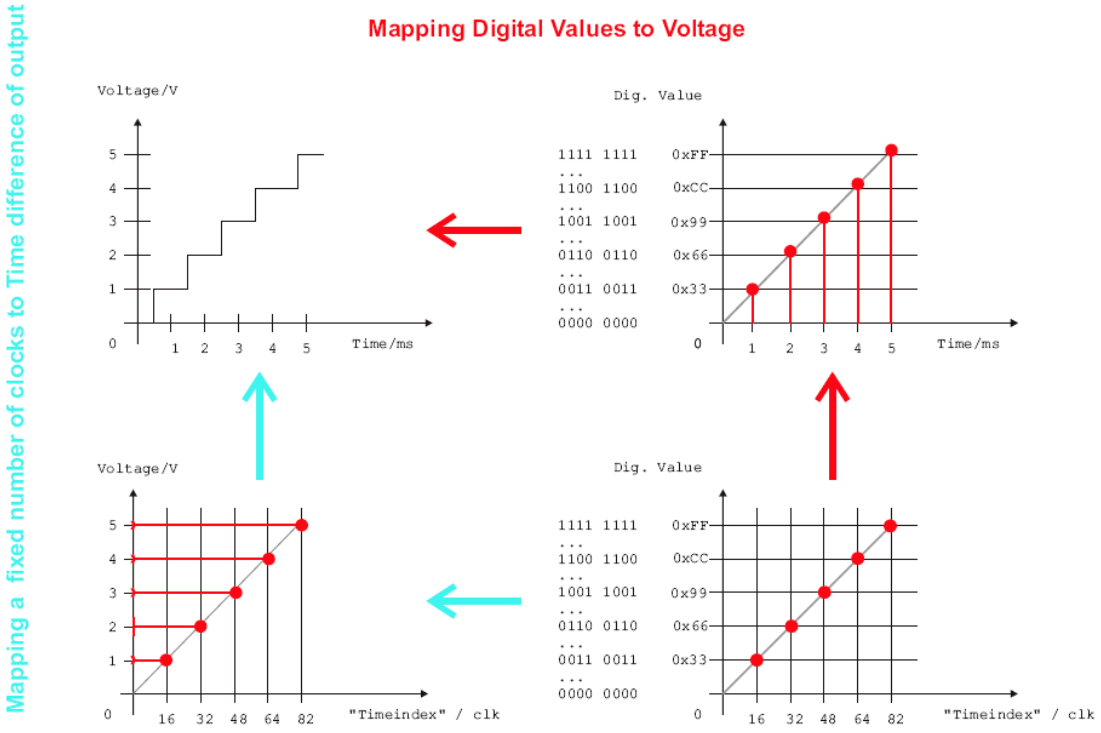


Abbildung von ganzzahligen Werten und diskreter Zeitpunkte auf reelle Analogwerte



Analog-to-Digital Conversion

- Amplitude: Quantisierung

  i.d.R. lineare Abbildung auf 2^n Werte

  U_{max} = Fullscale Value: $U_{max}/2^n = 1$ LSB, U_{max} häufig auch U_{ref}
- Zeit: Diskretisierung

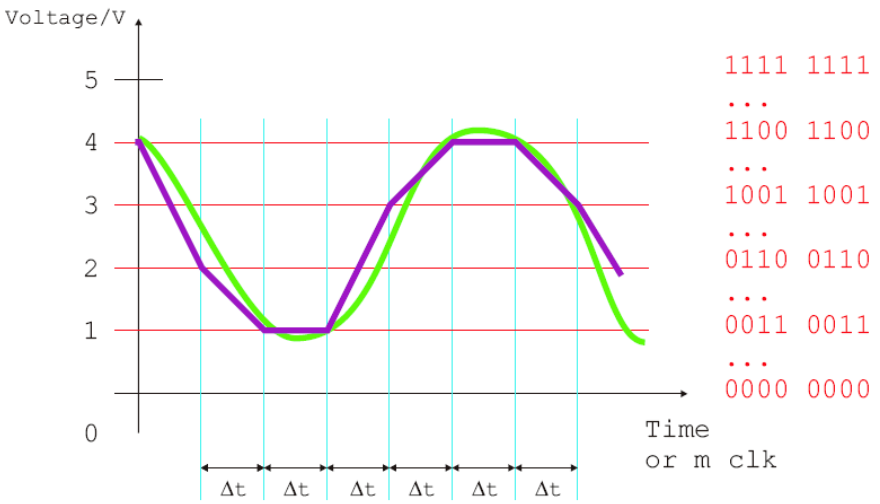
  i.d.R. äquidistante Abbildung im zeitlichen Abstand Δt auf potentiell ∞ Abtastzeitpunkte

  Δt = Sampling- oder Conversion Time, $\frac{1}{\Delta t}$ Samplingrate
- Die wichtigsten Parameter eines ADC bilden stets ein Paar:

1) n Bit Resolution [ohne Einheit]

2) $\frac{1}{\Delta t}$ Samplingrate [in (k,M)Samples p. Second oder Hz, kHz, MHz]

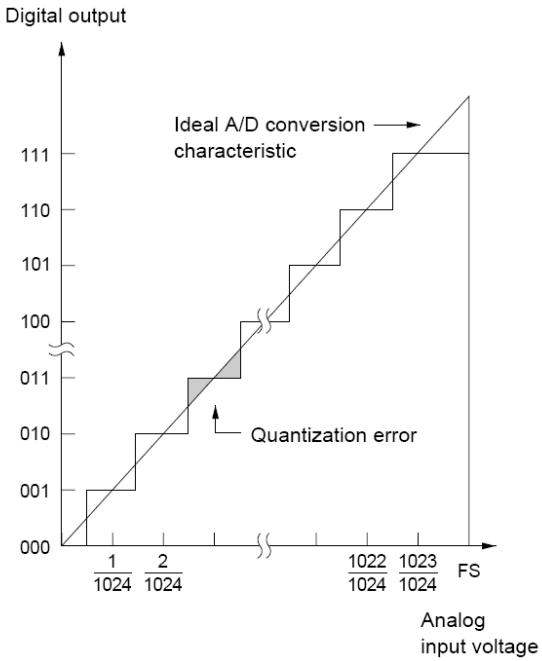
Fehler der Abbildungen: Quantisierungs- und Diskretisierungsfehler



Time: Sampling => Most algorithms require equidistant sampling
=> Array indexing fits to an equistant "timeindex"

Values: Quantization => Most A/D converters support 2^n equistant steps (n typ. 8-20)
=> often fitted to datatypes (byte, word ...)

Kleinstmöglicher systematischer Quantisierungs-Fehler = 1/2 LSB



Quiz

- Geben Sie die Quantisierung / Samplingrate von ADC's an:
- Telefon -Sprachübertragung:

  Resolution =

  Samplingrate =

  Frequenzband =
- High-End Audio - Wiedergabe:

  Resolution =

  Samplingrate =

  Frequenzband =
- Digital Oszilloskop:

  Resolution =

  Samplingrate =

  Frequenzband =

Quiz

- Geben Sie die Quantisierung / Samplingrate von ADC's an:
- Telefon-Sprachübertragung:

  Resolution = 8 Bit

  Samplingrate = 8 kSPS

  Frequenzband = typ. 100Hz-3kHz
- High-End-Audio - Wiedergabe:

  Resolution = 16 Bit, max. 24 Bit

  Samplingrate = 44 oder 48 kSPS (max. 96 kSPS)

  Frequenzband = 20Hz- ca. 20 kHz Bandbreite
- Digital Oszilloskop:

  Resolution = 8 Bit (ohne zus. Signalverarbeitung)

  Samplingrate =mehrere 100 MSPS (max. 1-5 GSPS)

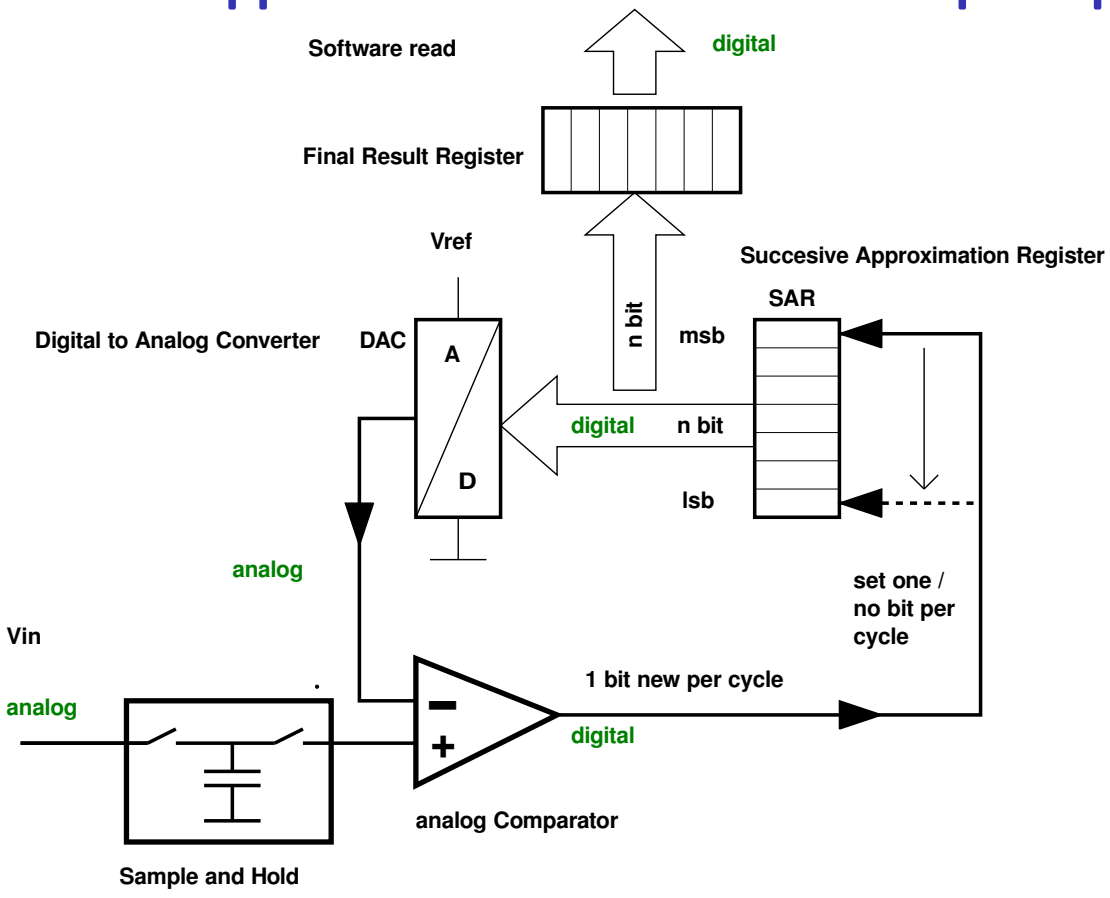
  Frequenzband = mHz bis typ. mehrere hundert MHz (max. wenige GHz)

Typische Umsetzungsmethoden und Parameter

Method	Direct / Pipelined /Flash	Successive Approximation	Sigma-Delta
resolution	low (8-14 bit)	average (8-18 bit)	high (16-24 bit)
conversion rate	very high up to 500 MHz	medium to high up to 1 MHz	low to medium 1 Hz - 100 kHz
power dissipation	very high up to 3 W	medium to high 1-200mW	low to medium 0.2 - 10m

The Data Conversion Handbook, edited by Walt Kester (Newnes, 2005), free download
http://www.analog.com/library/analogdialogue/archives/39-06/data_conversion_handbook.html

Successive Approximation als ADC Grundprinzip



Komponenten des Successive Approximation ADC

- Das Sample-and-Hold-Glied hält den Spannungswert V_{in} während der Umsetzung fest. Spannungsverlust des Kondensators $\ll V_{lsb}$ (dt. U_{lsb})
- Der Comparator gibt eine logische '1' aus, wenn $V_{DAC} < V_{in}$

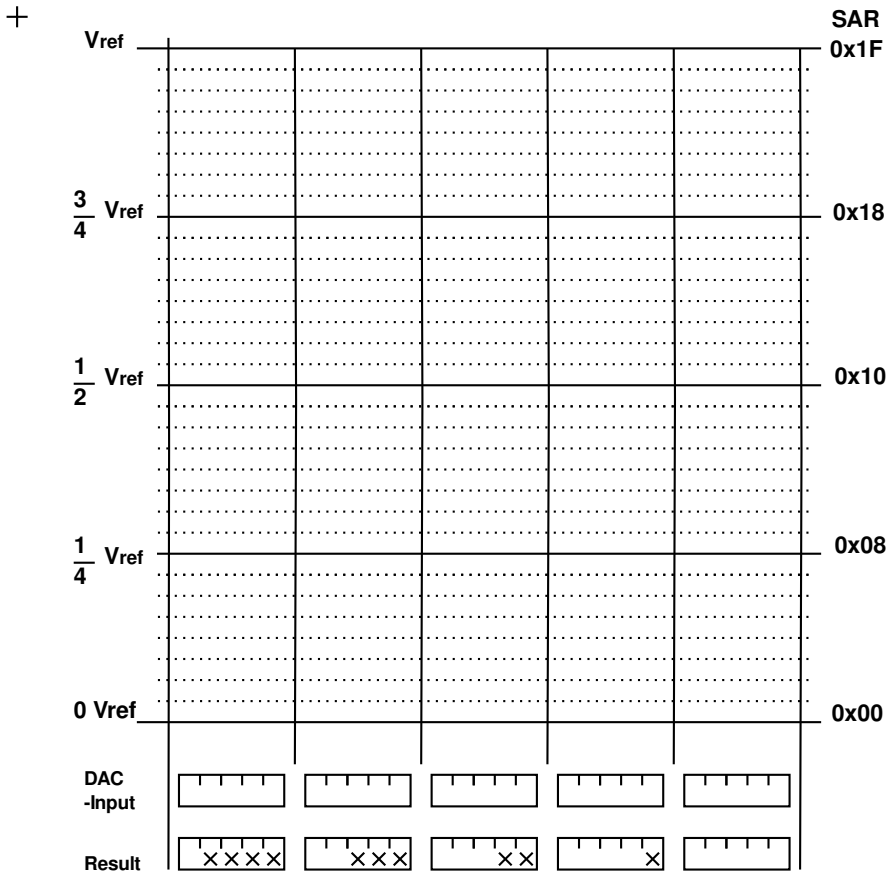
  Er gibt eine logische '0' aus, wenn $V_{DAC} > V_{in}$.

  $V_{DAC} = V_{in}$ ist praktisch ausgeschlossen, der Comparator entscheidet immer.
- Das Successive Approximation Register = SAR gibt den Eingangswert des DAC vor. Am Ende wird das Umsetzungsergebnissaus dem SAR in das Final-Result-Register ausgelesen.
- Der Digital Analog Converter setzt den SAR um. Er bezieht sich auf V_ref.
- Die Digitale Logik ändert in n Schritten die n Bitwerte im SAR einzeln nacheinander. MSB $\Rightarrow$ LSB.

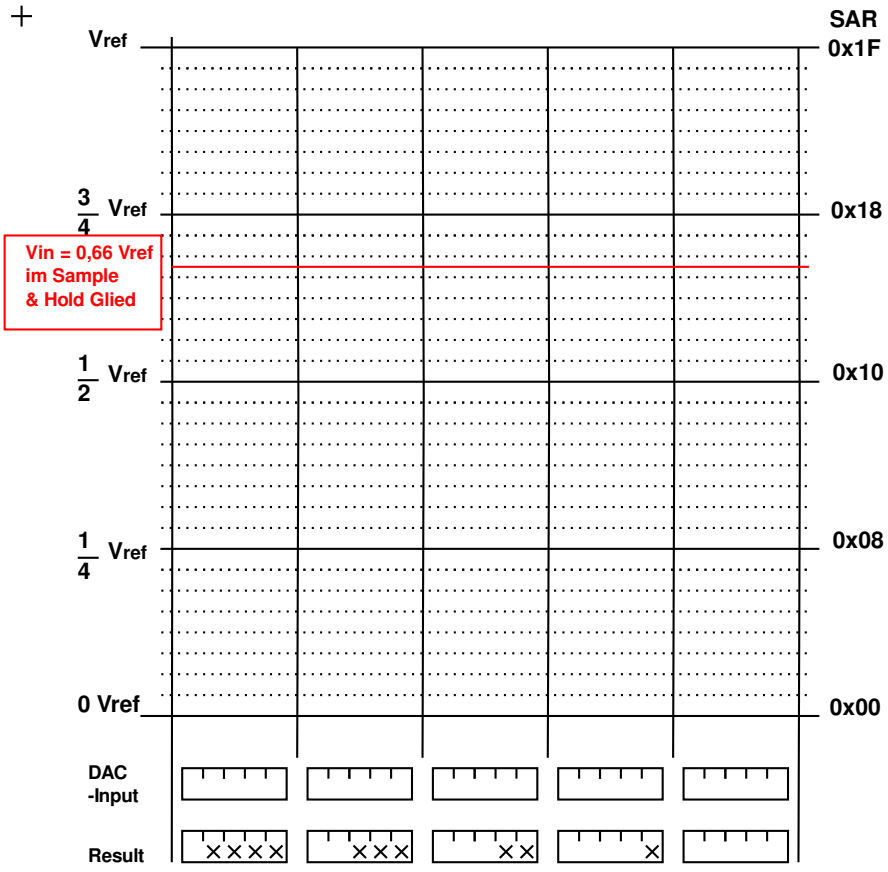
  Anfangs-Vorgabe an den DAC MSB = '1' Rest '0'.

  Der Bit-Eintrag im SAR nach jedem Schritt ist das Ergebnis des Comparators.

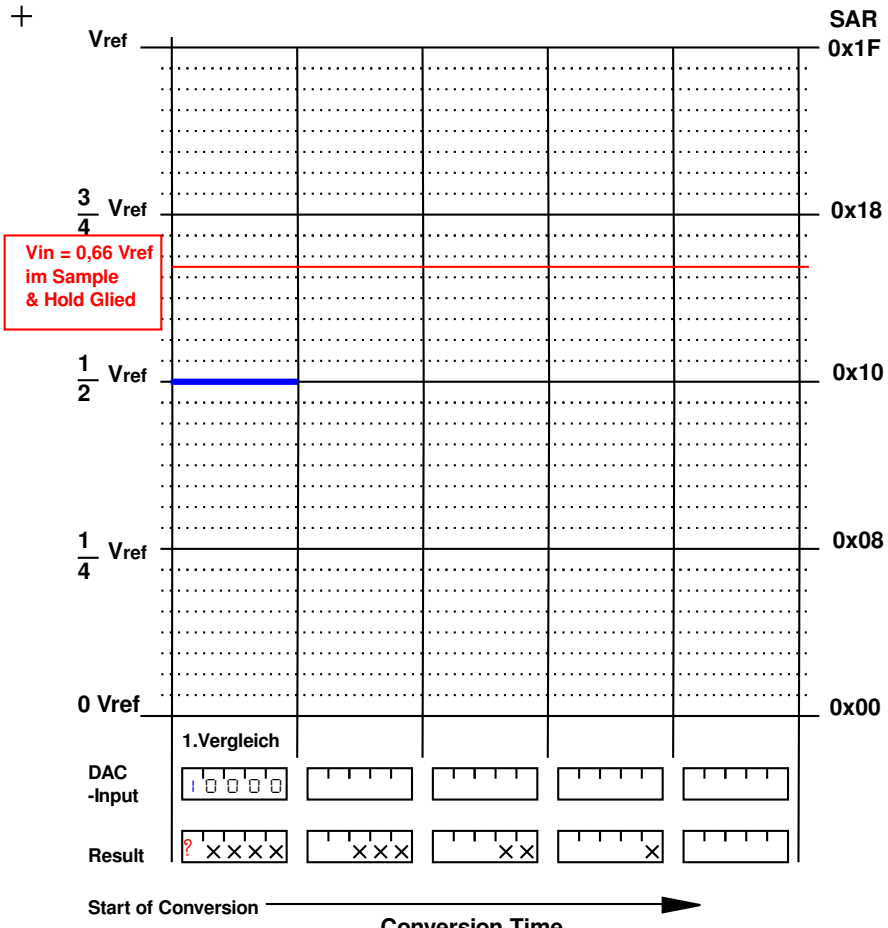
SAR-Conversion Beispiel I



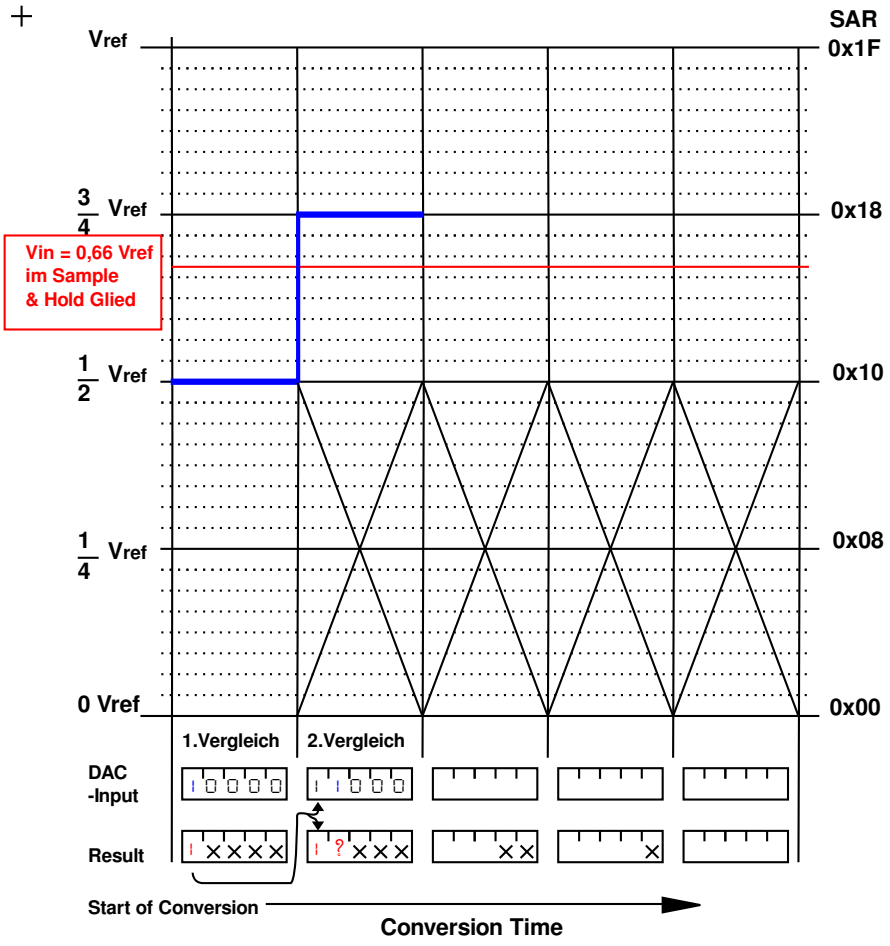
SAR-Conversion Beispiel II



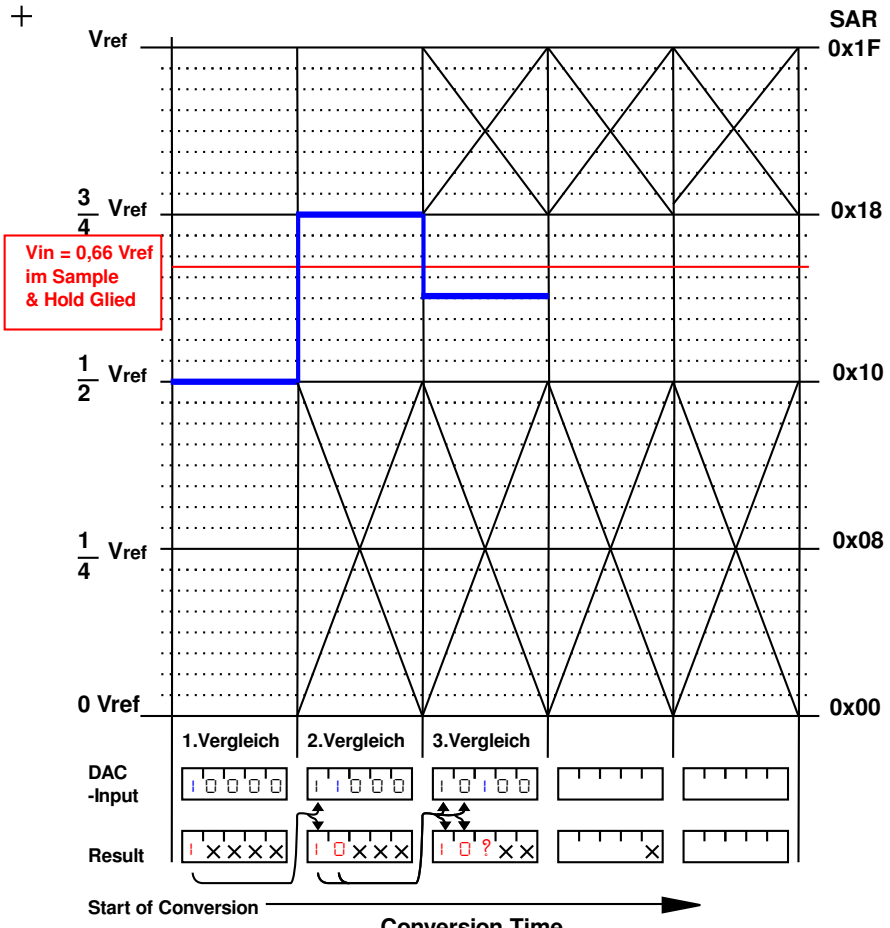
SAR-Conversion Beispiel III



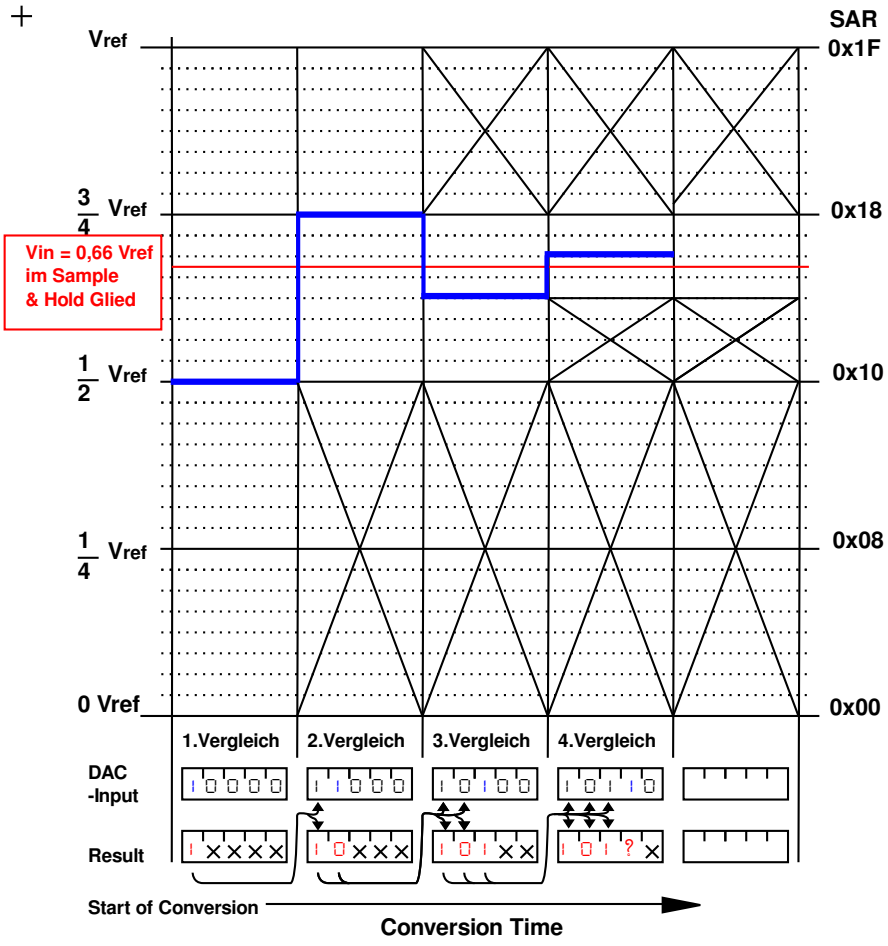
SAR-Conversion Beispiel IV



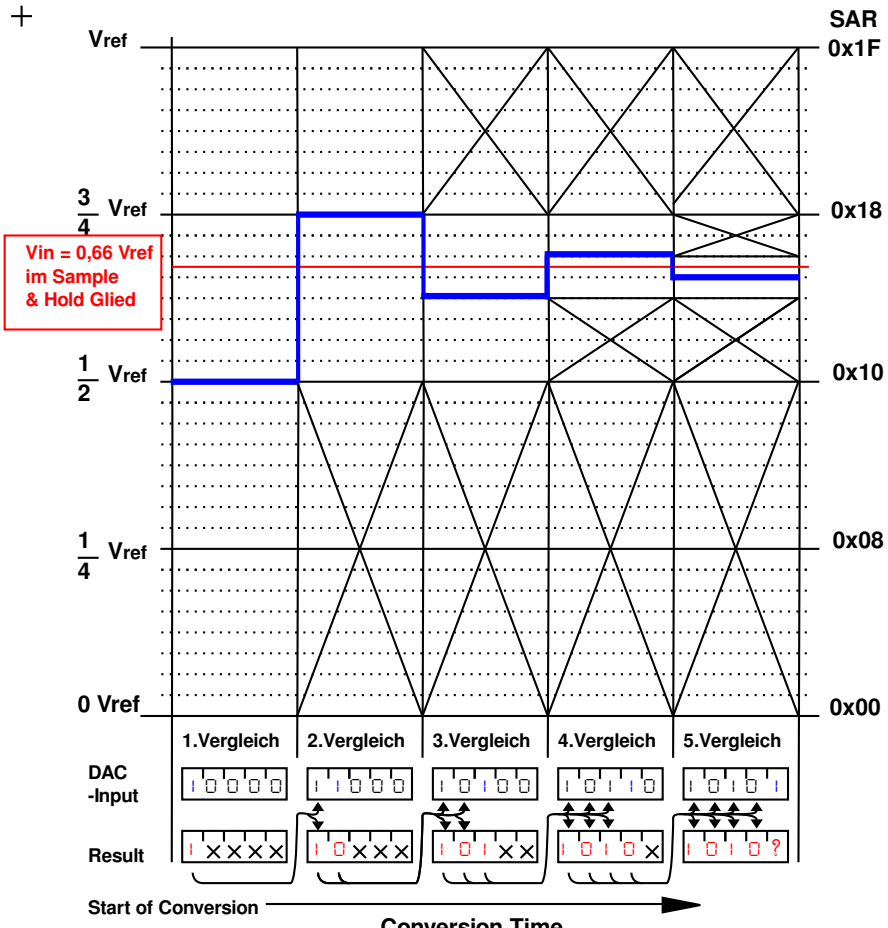
SAR-Conversion Beispiel V



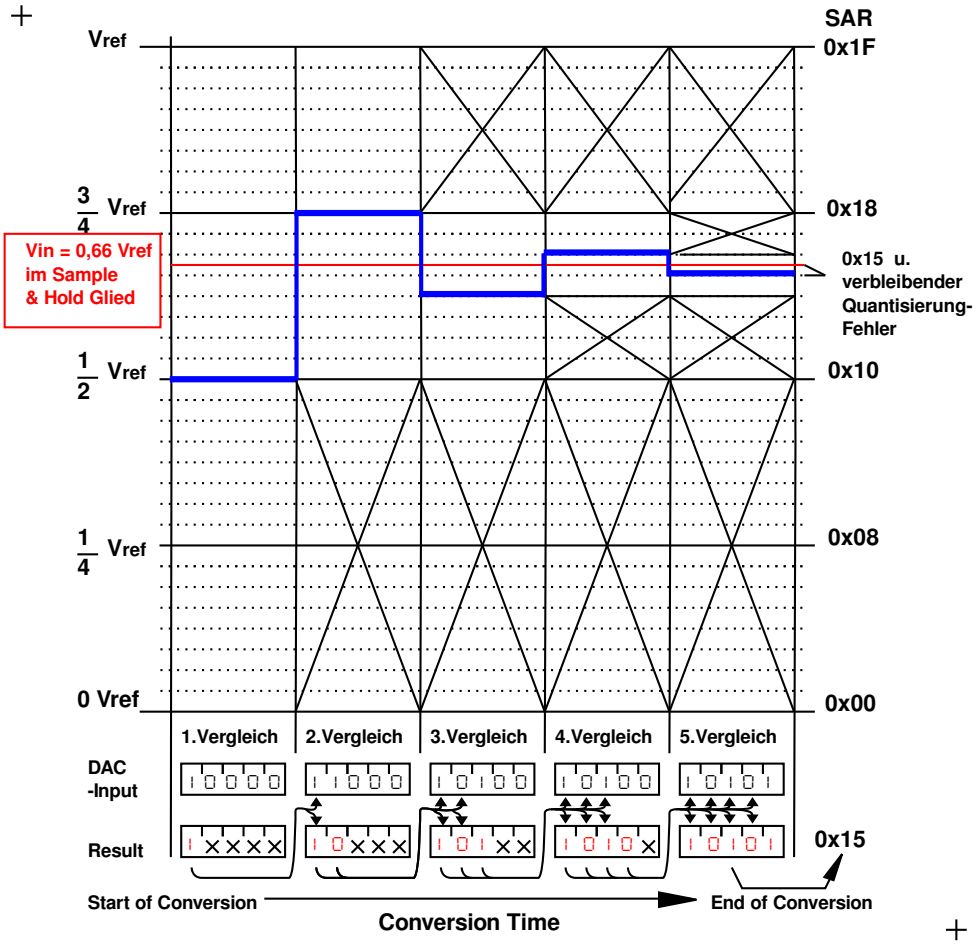
SAR-Conversion Beispiel VI



SAR-Conversion Beispiel VII



SAR-Conversion Beispiel VIII

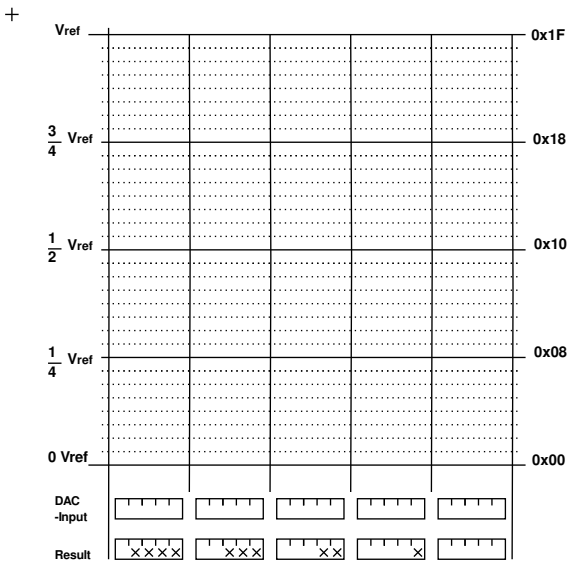


SAR-Conversion 5Bit Auflösung Beispiel

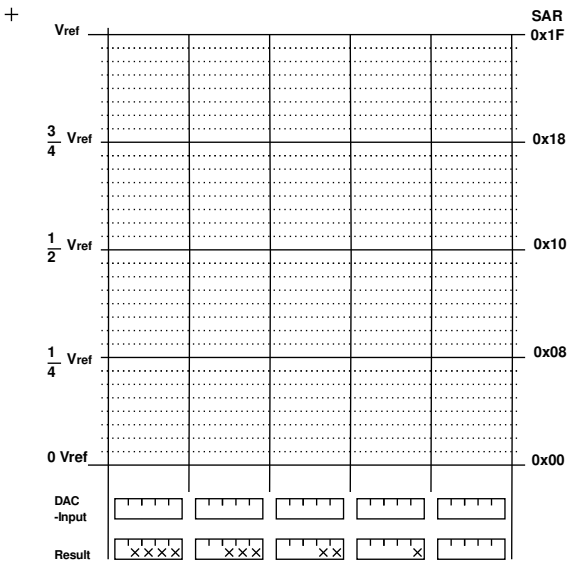
5 Bit ADC	Vref/32	Vref/32(Dec)	Stufe	SAR (Hex)	SAR(Bin)
Vref		1,00000			
Vref-V(LSB)	31/32	0,96875	31	0x1F	11111
	30/32	0,93750	30	0x1E	11110
	29/32	0,90625	29	0x1D	11101
	28/32	0,87500	28	0x1C	11100
	27/32	0,84375	27	0x1B	11011
	26/32	0,81250	26	0x1A	11010
	25/32	0,78125	25	0x19	11001
3/4*Vref-Vlbs	24/32	0,75000	24	0x18	11000
	23/32	0,71875	23	0x17	10111
	22/32	0,68750	22	0x16	10110
	21/32	0,65625	21	0x15	10101
	20/32	0,62500	20	0x14	10100
	19/32	0,59375	19	0x13	10011
	18/32	0,56250	18	0x12	10010
	17/32	0,53125	17	0x11	10001
1/2*Vref-Vlsb	16/32	0,50000	16	0x10	10000
	15/32	0,46875	15	0x0F	01111
	14/32	0,43750	14	0x0E	01110
	13/32	0,40625	13	0x0D	01101
	12/32	0,37500	12	0x0C	01100
	11/32	0,34375	11	0x0B	01011
	10/32	0,31250	10	0x0A	01010
	9/32	0,28125	9	0x09	01001
1/4*Vref-Vlsb	8/32	0,25000	8	0x08	01000
	7/32	0,21875	7	0x07	00111
	6/32	0,18750	6	0x06	00110
	5/32	0,15625	5	0x05	00101
	4/32	0,12500	4	0x04	00100
	3/32	0,09375	3	0x03	00011
	2/32	0,06250	2	0x02	00010
Vlsb	1/32	0,03125	1	0x01	00001
0	0/32	0,00000	0	0x00	00000

SAR-Conversion Arbeitsblatt

Vin = V sample & hold = 0,16 * Vref



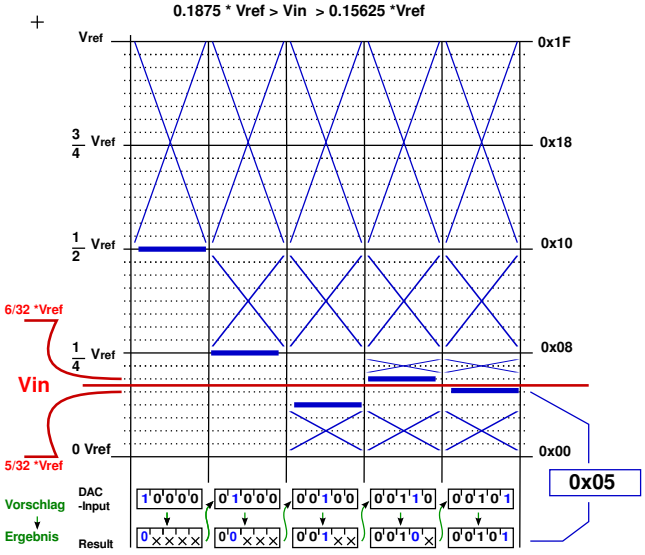
Vin = V sample & hold = 0,45 * Vref



SAR-Conversion Arbeitsblatt Lösung

Gegeben: Vin = V sample & hold = 0,16 * Vref

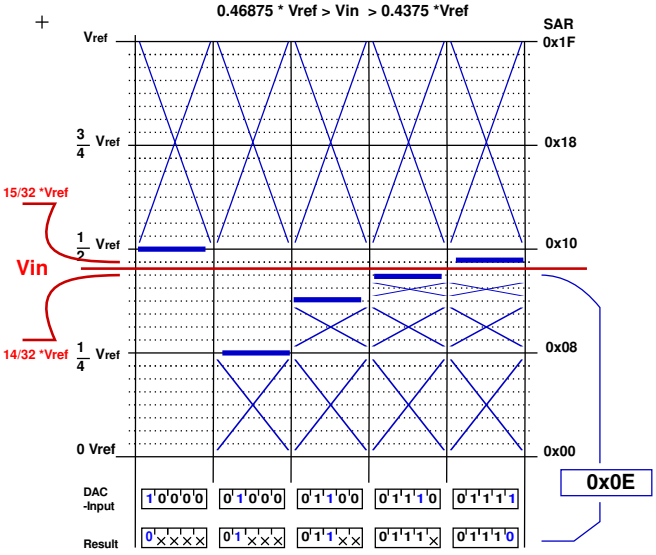
Wo ist Vref einzuzeichnen ?
6/32 *Vref = 0,1875 * Vref
5/32 *Vref = 0.15625 * Vref
0.1875 * Vref > Vin > 0.15625 *Vref



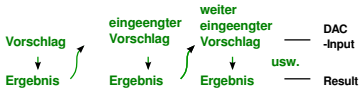
Antwort: Der Wert liegt über 0x05,
aber max 1 LSB *Vref = 1/32 *Vref darüber
also zwischen 0x05 und 0x06

Vin = V sample & hold = 0,45 * Vref

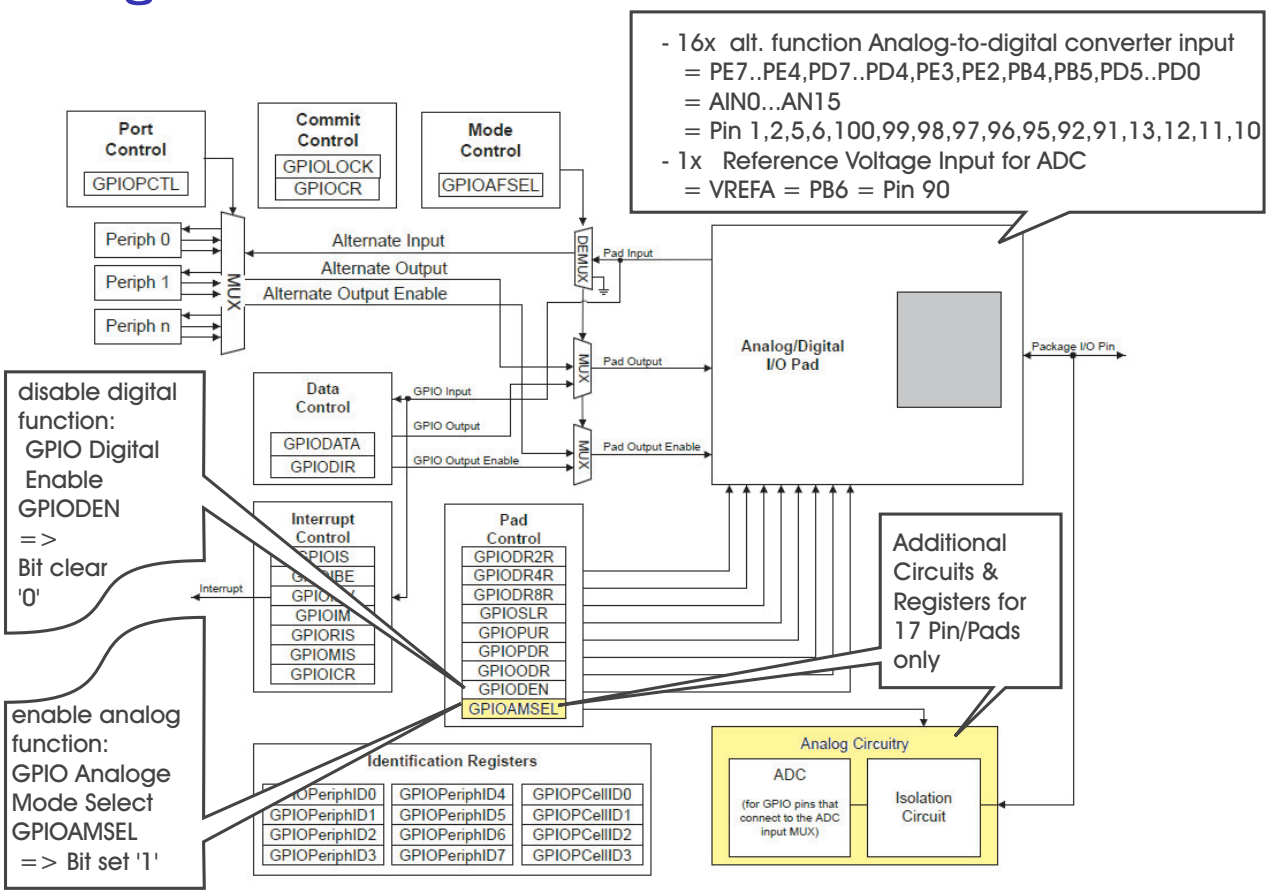
Wo ist Vref einzuzeichnen ?
15/32 *Vref = 0,46875 * Vref
5/32 *Vref = 0.4375 * Vref
0.46875 * Vref > Vin > 0.4375 *Vref



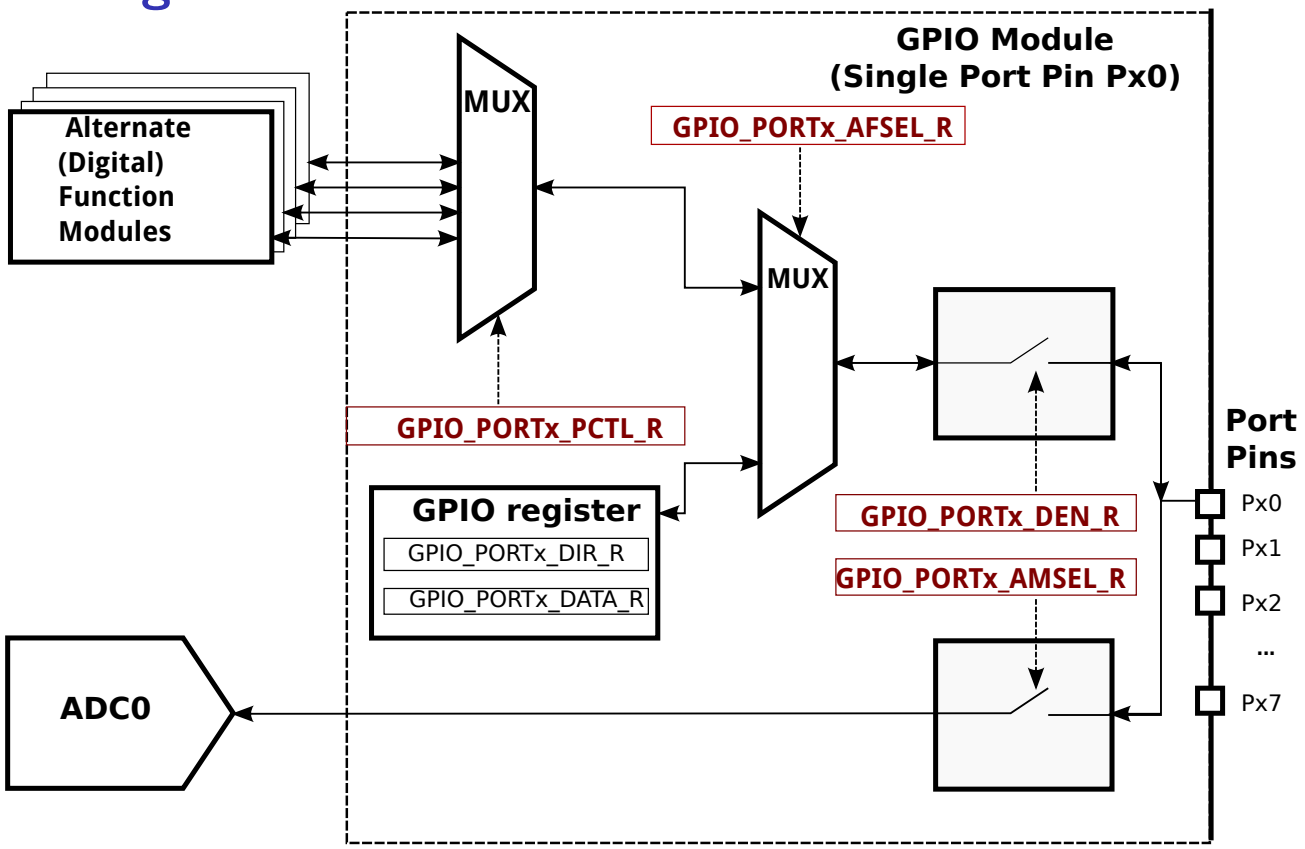
Antwort: Der Wert liegt über 0x0E,
aber max 1 LSB *Vref = 1/32 *Vref darüber
also zwischen 0x0E und 0x0F



Analog Pins



Analog Pin Function



Quelle: L. Leutelt, Vorlesungsunterlagen

ADC im TM4C1294

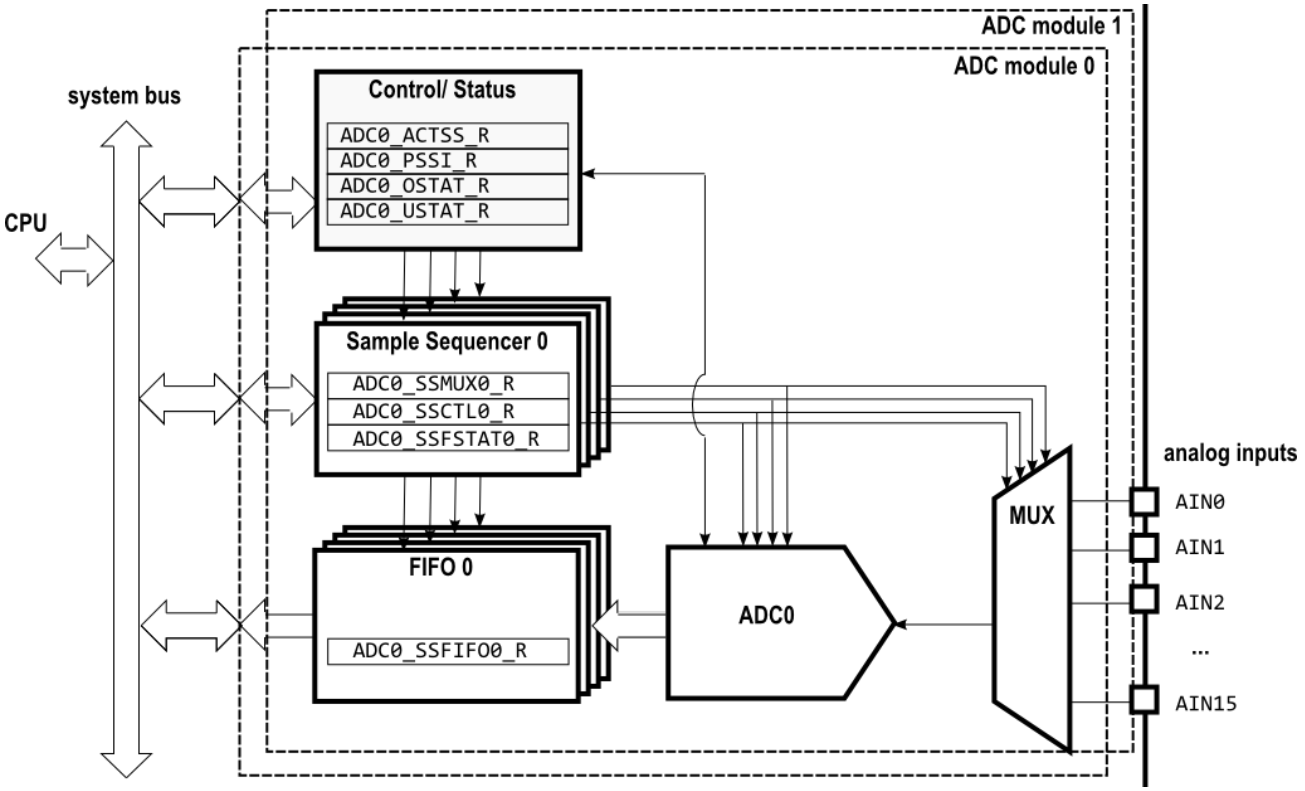
- Zwei separate ADC-Module ADC 0 und ADC 1
- 20 analoge Eingangsleitungen AIN0-AIN20- 2 im Labor vorbereitet an PE0 und PE1

  mit Multiplexer-Schaltungen zu den ADC Modulen ADC0 und ADC1
- Ausgabepuffer nach dem FIFO Prinzip (First In First Out) speichern die Ausgabe Werte bis zum Lesen durch SW
- Sample Sequencer steuern:

  welche Eingangsleitungen AINx vom ADC gelesen werden (max. 8)

  in welcher Reihenfolge die Eingangsleitungen AINx vom ADC gelesen werden
- Control/Status Blöcke steuern die Sample Sequencer

Blöcke des ADC



Quelle: L.Leutelt, Vorlesungsunterlagen

Sample Sequencer der ADC-Module

- Vier Sample Sequencer steuern die ausgelesenen analogen Eingangsleitungen.

Sequencer	Anzahl der Samples	Tiefe des FIFO
SS3	1	1
SS2	max. 4	4
SS1	max. 4	4
SS0	max. 8	8

- Die Konfiguration erfolgt in den Registern `ADCx_SSMUXn_R` und `ADCx_SSCCTLn_R` (x=0,1 und n=0,1,2,3)

  Beispielkonfiguration für SS0:

Sample 7	Sample 6	Sample 5	Sample 4	Sample 3	Sample 2	Sample 1	Sample 0
-	-	AN0	AIN3	AN2	AN0	AN14	AN11

Hinweise: Sequenzen können vorzeitig enden, Wiederholungen sind erlaubt (Hier: AN0 wird wiederholt, Sample 7 und 6 nicht in der Sequenz)

FIFO Zwischenspeicher der ADC-Module

- Vier FIFO werden durch Sample Sequencer gesteuert.

  Beispiel einer FIFO-Belegung gemäß vorheriger Konfiguration für SS0:

FIFO Pos.7	FIFO Pos.6	FIFO Pos.5	FIFO Pos.4	FIFO Pos.3	FIFO Pos.2	FIFO Pos.1	FIFO Pos.0
-	-	AIN0	AIN3	AN2	AIN0	AIN14	AIN11

  FIFO Pos.0 ist hier zu zuerst beschrieben worden und wird zuerst gelesen.

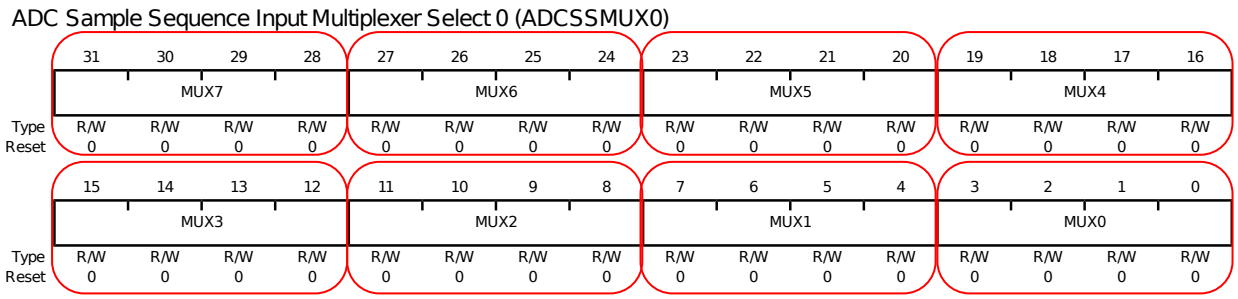
- Die Software liest aus dem Register `ADCx_SSFIFOn_R` (x=0,1 und n=0,1,2,3)

  die Werte aus dem FIFO nacheinander aus.

- Jedes Lesen durch SW entfernt einen gültigen Wert aus dem FIFO.
- Jede abgeschlossene AD-Conversion schreibt einen neuen gültigen Wert in den FIFO.

Registerkonfiguration der Sample Sequencer (I): `ADCx_SSMUXn_R`(x=0,1 ; n=0,1,2,3)

- Die in die Sequenz ausgewählten Eingangsleitungen werden in die Bitfelder des Registers eingetragen.



MUX0 wird zuerst genutzt, MUX7 oder eine als letzte bestimmte Leitung sind das letzte Input in der Sequenz. Die äquivalenten Register für SS1, SS2, SS3 haben gleichartige Bitfelder, jedoch weniger.

Registerkonfiguration der Sample Sequencer (II): ADCx_SSCTLn_R(x=0,1; n=0,1,2,3)

- Das Ende der Sequenz aus ausgewählten Eingangsleitungen wird durch Setzen des Bit ENDm(m=0,..,7) in den 8 Bitfelder des Registers eingetragen.

ADC Sample Sequence Control 0 (ADCSSCTL0)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	TS7	IE7	END7	D7	TS6	IE6	END6	D6	TS5	IE5	END5	D5	TS4	IE4	END4	D4
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TS3	IE3	END3	D3	TS2	IE2	END2	D2	TS1	IE1	END1	D1	TS0	IE0	END0	D0
Type	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Weitere Bits siehe Datasheet S. 654

Ausgabedaten Register des FIFO der Sample Sequencer: ADCx_SSFIFOn_R(x=0,1; n=0,1,2,3)

- Das Bitfeld DATA enthält DATEN (10bit), die der FIFO zur Ausgabe bereithält, falls dieses der Fall ist.
- Das Register ist Read-Only und Read-Sensitiv, nach jedem Lesen wird das Register automatisch von HW neu geladen, wenn Werte im FIFO sind.

ADC Sample Sequence Result FIFO n (ADCSSFIFOn)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

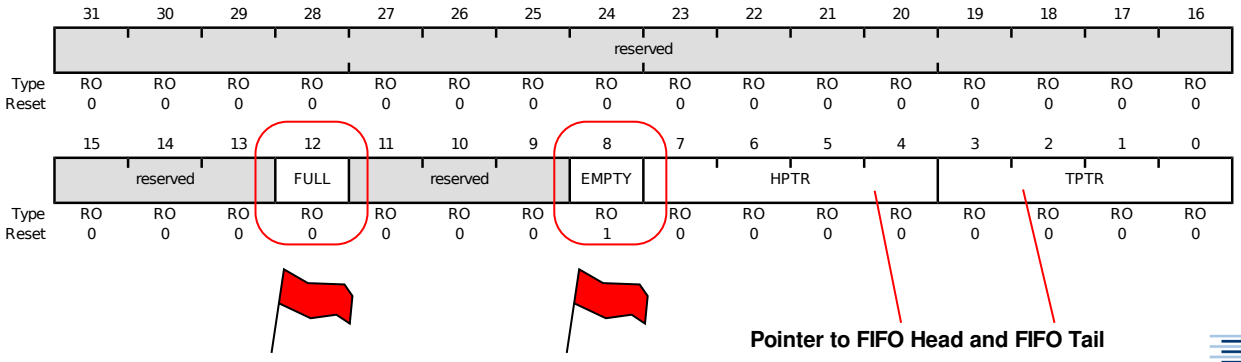
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved						DATA									
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-

Read-Only and Read-Sensitive: Automatic Update from FIFO

Flag-Register des FIFO der Sample Sequencer: ADCx_SSFSTATn_R(x=0,1; n=0,1,2,3)

- Das Register enthält die Flags des FIFO Zwischenspeichers.
- Das Flag Empty (= 1 !) zeigt einen leeren FIFO an.
- Die SW kann bei Empty = 0 Daten auslesen.
- Das Flag FULL (= 1 !) zeigt einen vollständig gefüllten FIFO an.
- Die SW muss auslesen, damit wieder (gültige) Daten eingespeichert werden können. Sonst Überlauffehler.
- Head und Tail sind die internen Zeiger des FIFO.

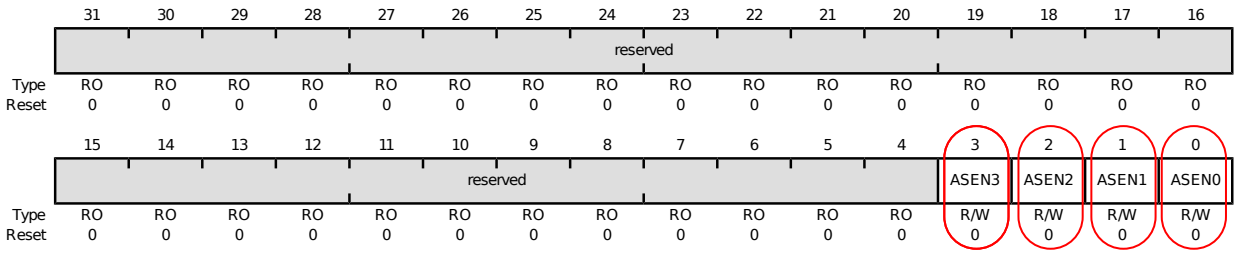
ADC Sample Sequence FIFO 0 Status (ADCSSFSTAT0)



Auswahl des (der) aktiven Sample Sequencer: ADCx_ACTSSn_R(x=0,1; n=0,1,2,3)

- In diesem Register ist das Freigeben (active) oder Abschalten (not active) der vier Sample Sequencer möglich.
- Während der Konfiguration müssen die Sample Sequencer deaktiviert sein.

ADC Active Sample Sequencer (ADCACTSS)



Starten der ADC Conversion des (der) Sample Sequencer: `ADCx_PSSIn_R`(x=0,1; n=0,1,2,3)

- In diesem Register werden die ADC-Funktion für die Sample Sequencer gestartet (to initiate).
- Die Bits werden nach dem Start sofort automatisch von der HW gelöscht.
- Das Bit GSYNC erlaubt den gleichzeitigen Start von Sequenzern im ADC0 und ADC1 Modul

ADC Processor Sample Sequence Initiate (ADCPSSI)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	GSYNC	reserved			SYNCWAIT	reserved										
Type	R/W	RO	RO	RO	R/W	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved												SS3	SS2	SS1	SS0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	WO	WO	WO	WO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-

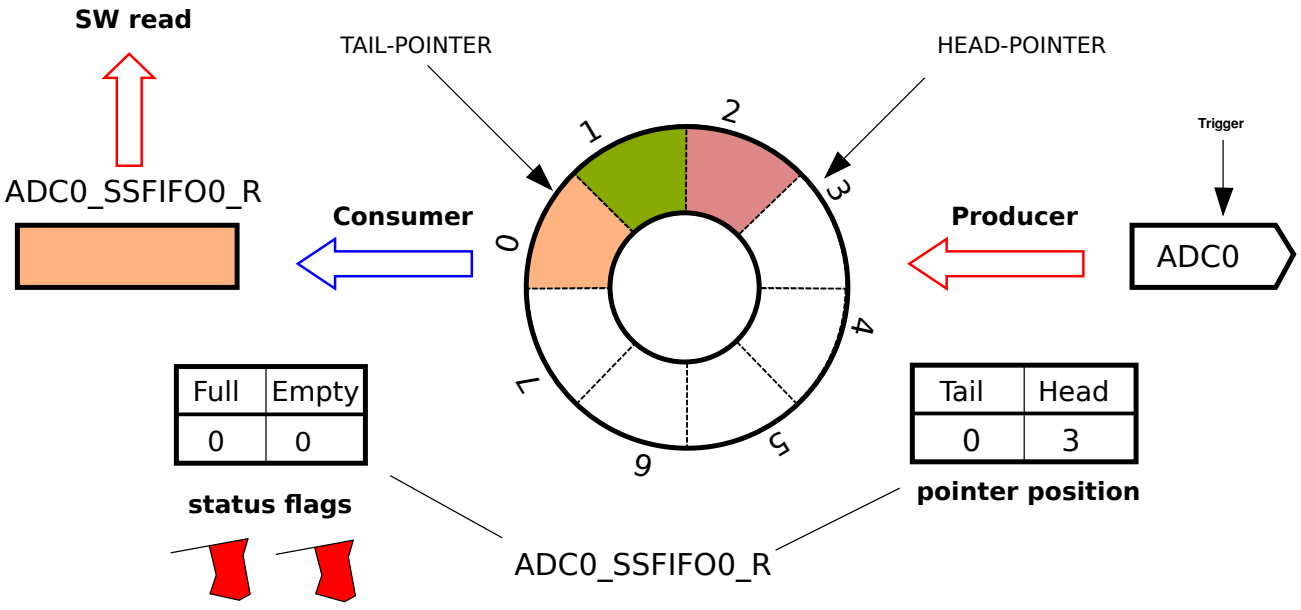
Simple ADC Example 1 of 2

```
1  // Simple ADC Testprogram: one channel + single conversion in a SW loop
2  //-----------------------------------------------------------------------------/
3  #include <stdint.h>
4  #include "stdio.h"
5  #include "inc/tm4c1294ncpdt.h"
6
7  #define ADC_VREF    3300.0  // voltage on V_REF+ pin in mV
8  #define V_LSB       (ADC_VREF / 4095)   // V_LSB voltage in mV
9  #define V_COEFF     ((4.70 + 9.137) / 9.137 * V_LSB)
10 // LSB voltage in mV, since resistive voltage divider is used,
11 // ADC input impedance assumed as constant => data sheet 1862ff.
12
13 int main(void) {
14  unsigned int ADCoutput, wt;
15
16  // Clock switch on AIN3 of ADC0  ... Pin is PE(5)
17 SYSCTL_RCGCGPIO_R  |= 0x00000010; // Clock Port E enable
18 SYSCTL_RCGCADC_R   |= 0x1; wt++;// Clock ADC0 enable  set
19
20 // Magic code for start the ADC Clocking
21 // => see tm4c1294ncpdt Datasheet, 15.3.2.7 ADC Module Clocking
22  SYSCTL_PLLFREQ0_R |= SYSCTL_PLLFREQ0_PLLPWR;   // power on the PLL
23  while(!(SYSCTL_PLLSTAT_R & SYSCTL_PLLSTAT_LOCK)); // wait till PLL has locked
24  ADC0_CC_R =  0x01 ;    wt++; // select PIOSC (internal RCOsc) as ADC analog clock
25  SYSCTL_PLLFREQ0_R &= ~SYSCTL_PLLFREQ0_PLLPWR;  // power off the PLL (s. above)
```


Simple ADC Example 2 of 2

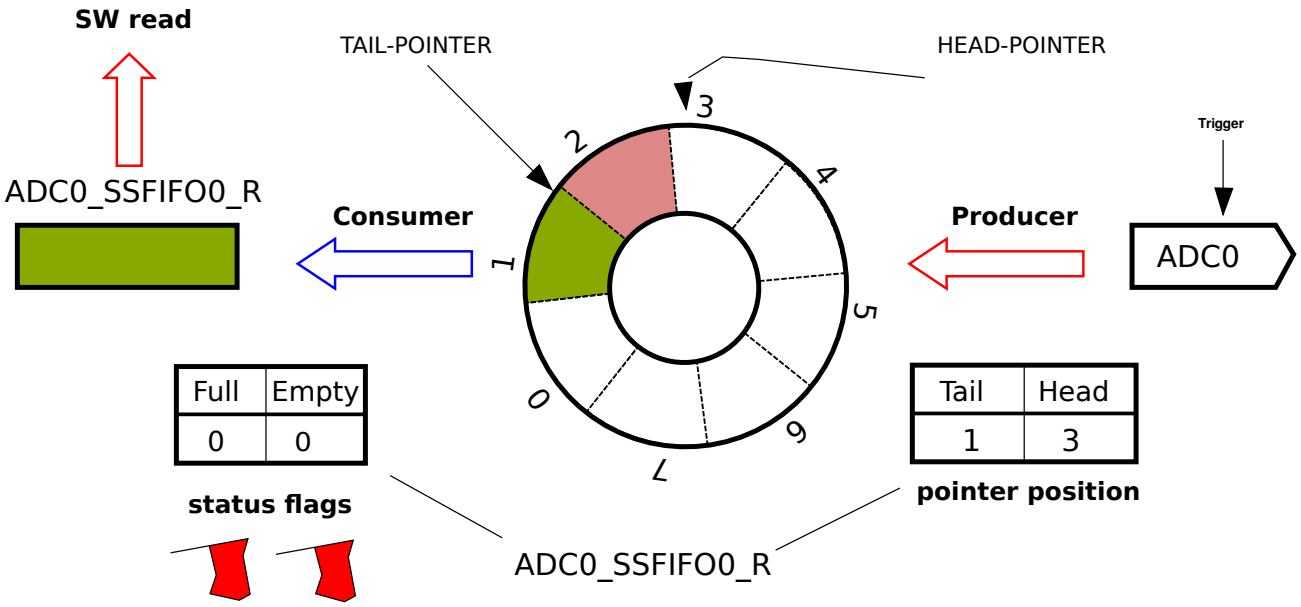
```
26 // end of magic code ...
27
28  // Prepare Port Pin PE0 as AIN3
29  GPIO_PORTE_AHB_AFSEL_R |=0x01; // PE0 Alternative Pin Function enable
30  GPIO_PORTE_AHB_AMSEL_R |=0x01; // PE0 Analog Pin Function enable
31  GPIO_PORTE_AHB_DEN_R   &=~0x01; // PE0 Digital Pin Function DISABLE
32  GPIO_PORTE_AHB_DIR_R   &=~0x01; // Allow Input PE0
33
34  ADC0_ACTSS_R       &= ~0x0F; // disable all 4 sequencers of ADC 0
35  ADC0_SSMUX0_R  =0x00000003; // Sequencer 0 channel AIN3  only no mux
36  ADC0_SSCTL0_R |=0x00000002; // Set "END=0" sequence length 1 (one sample sequence)
37  ADC0_CTL_R     =0x0;        // V_REF = Vdda 3.3V ... if Bit0 is clear
38  ADC0_ACTSS_R  |= 0x01; // enable sequencer 0 of ADC 0
39  while(1)
40  {
41  ADC0_PSSI_R|=0x01; // Start ADC0
42  while(ADC0_SSFSTAT0_R & 0x000000100); // wait for FIFO (inverted) Flag "EMPTY = False"
43    ADCoutput=(unsigned long) ADC0_SSFIFO0_R; // Take avalue from FIFO output
44    // Calculate Output in mV with respect to voltage divider 5:3 in the Lab
45    printf("0x%3x=%4d␣(dec)␣==>␣%04d␣[mV]␣\n", ADCoutput, ADCoutput, (int) ( ADCoutput *
        V_COEFF + 0.5));
46  }
47 }
```


FIFO des ADC 'non empty' + read I



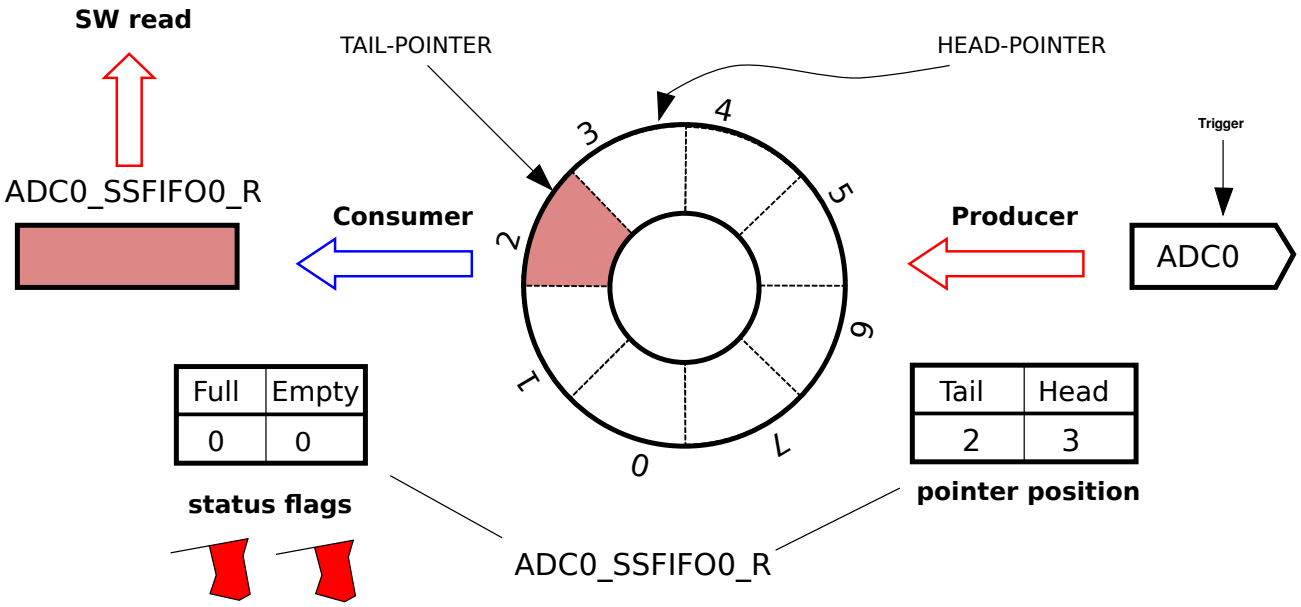
Quelle L. Leutelt modifiziert

FIFO des ADC 'non empty' + read II



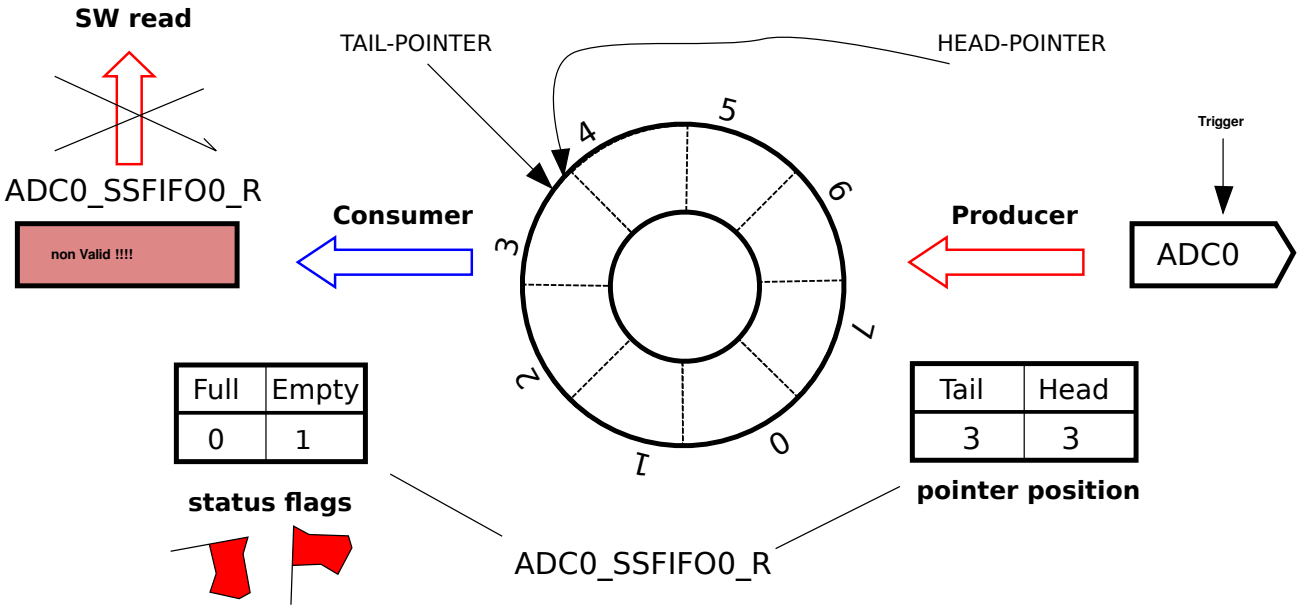
Quelle L. Leutelt modifiziert

FIFO des ADC 'non empty' + read III



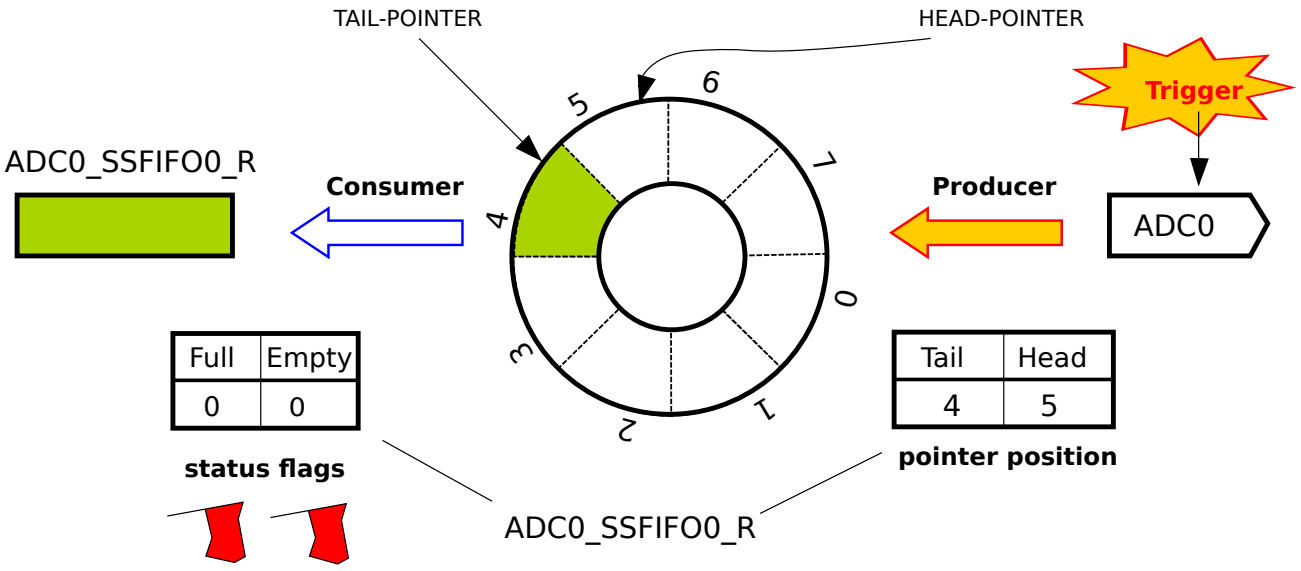
Quelle L. Leutelt modifiziert

FIFO des ADC 'empty' + read IV



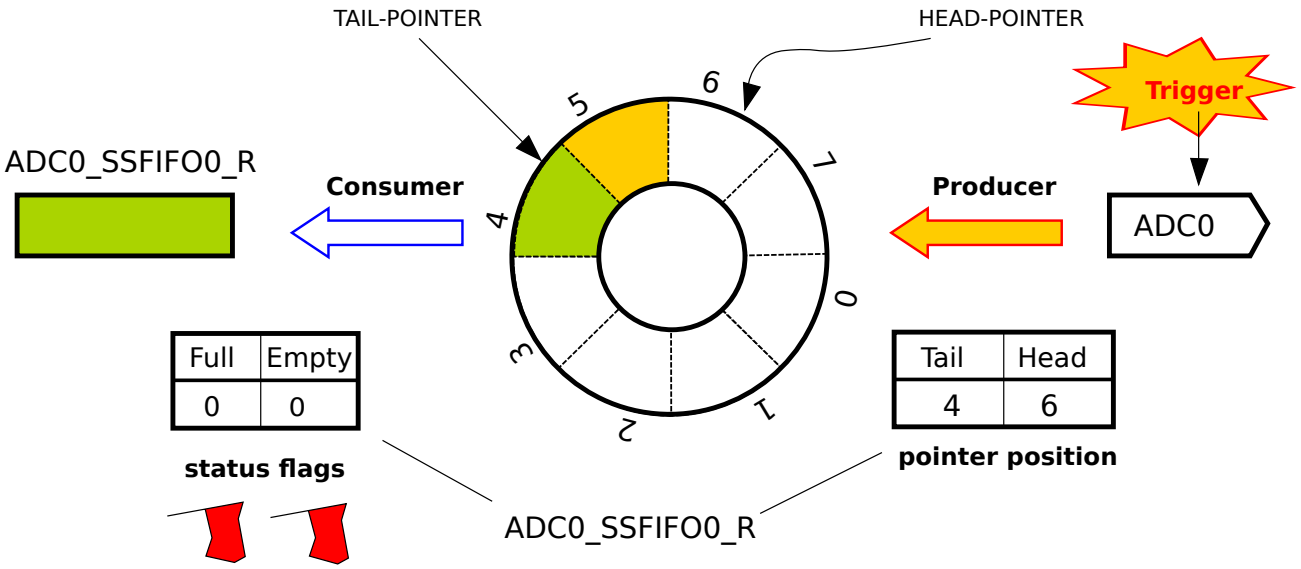
Quelle L. Leutelt modifiziert

FIFO des ADC 'non empty' + write V



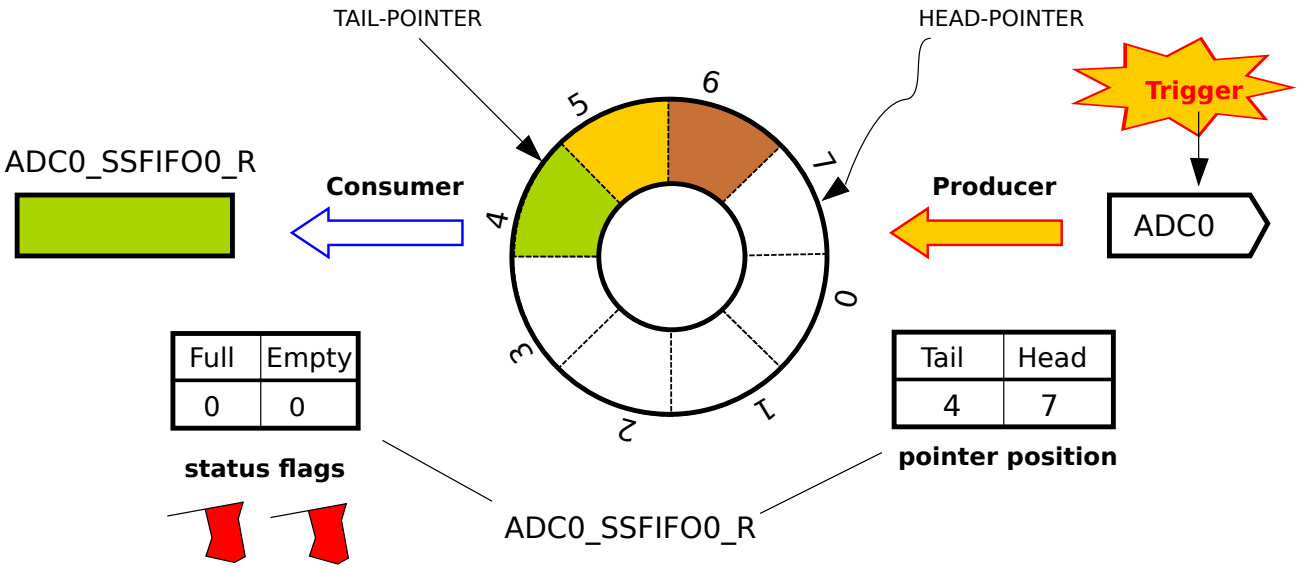
Quelle L. Leutelt modifiziert

FIFO des ADC 'non empty' + write VII



Quelle L. Leutelt modifiziert

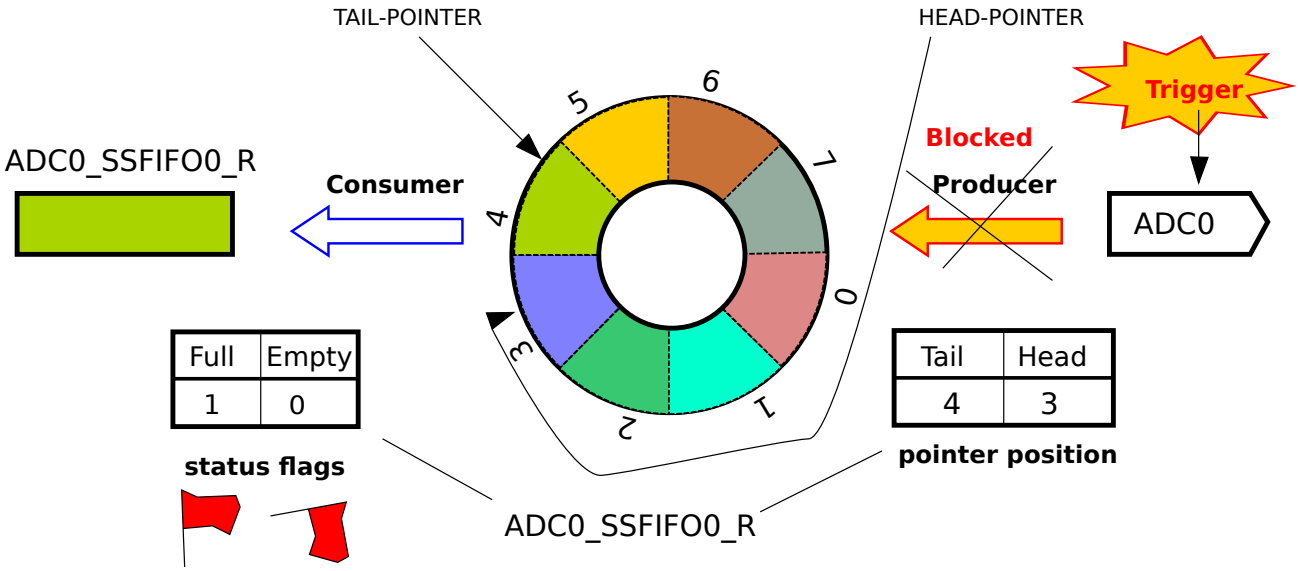
FIFO des ADC 'non empty' + write VIII



Quelle L. Leutelt modifiziert

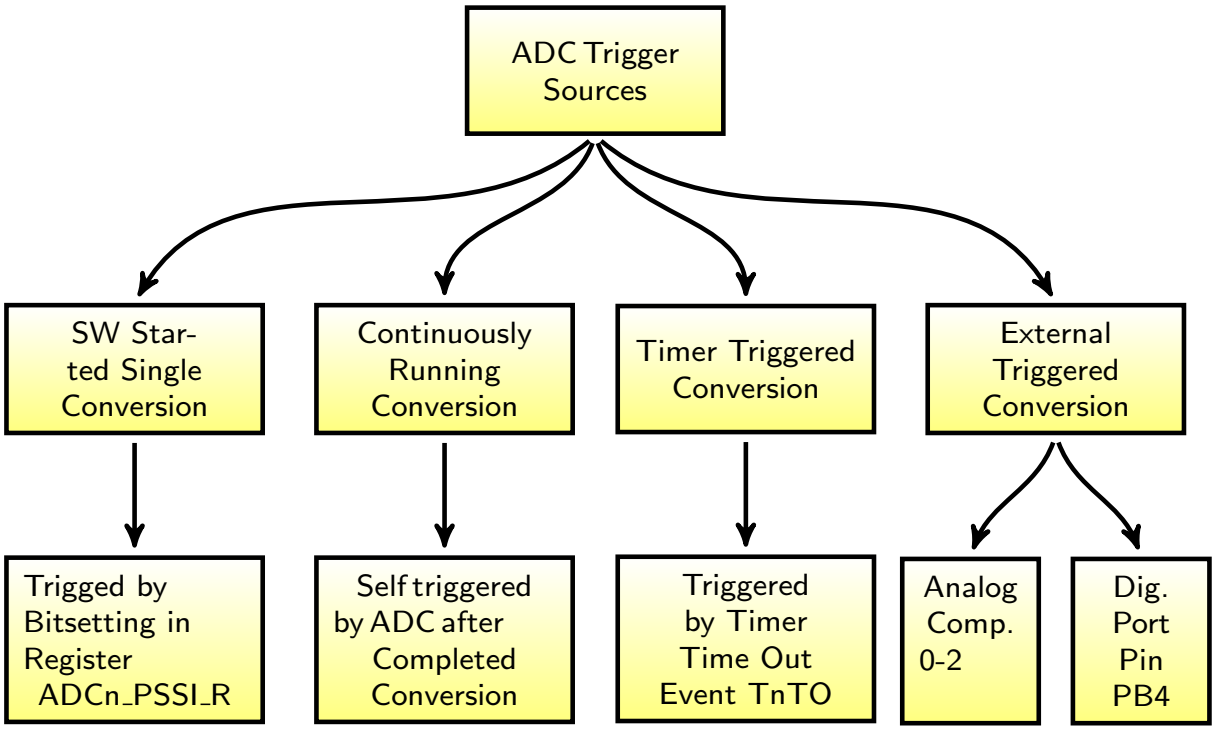
... usw. ...

FIFO des ADC 'full empty' + write blocked IX



Quelle L. Leutelt modifiziert

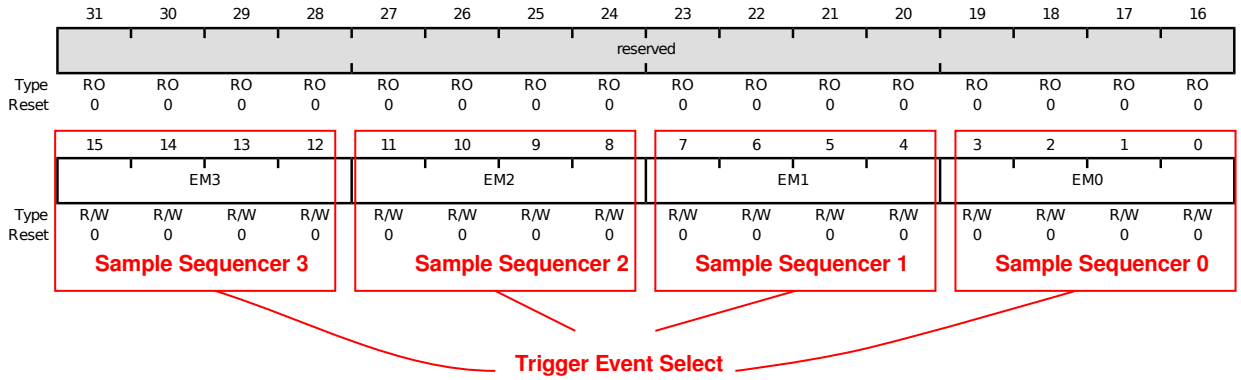
Trigger Quellen ADC



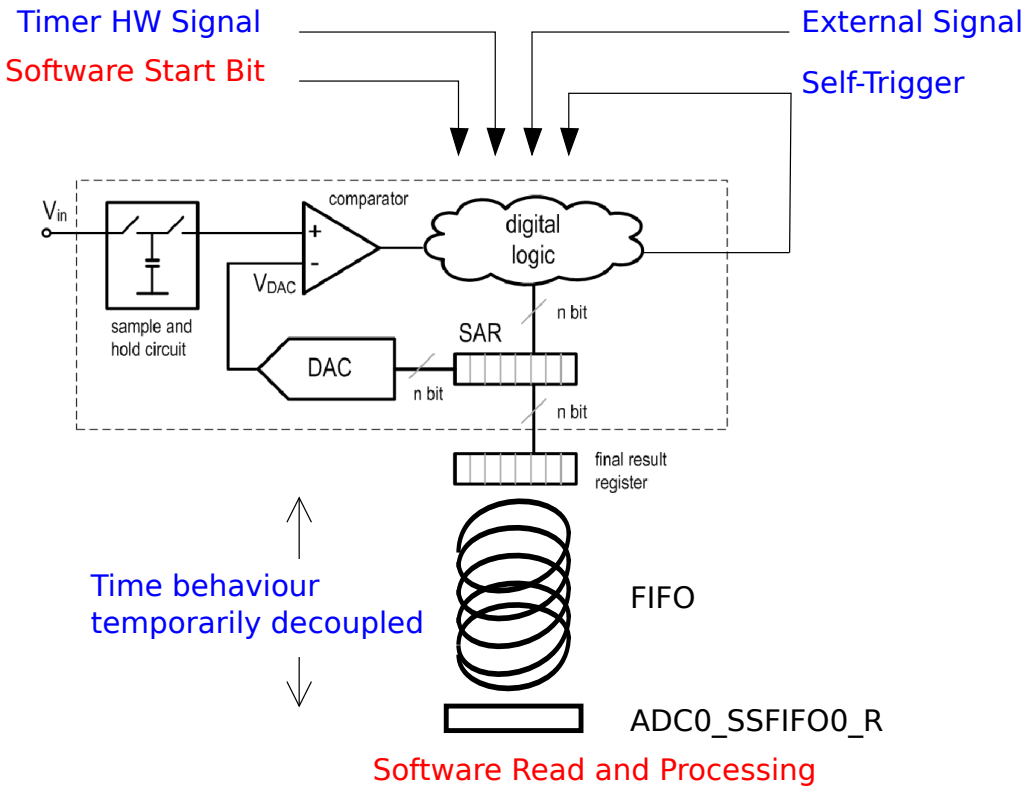
Auswahl der Trigger Quelle des (der) Sample Sequencer(s): ADCx_EMUX_R(x=0,1)

- In diesem Register wird die Trigger-Quelle gewählt.
- Für jeden Sequenzer SSn ein Bitfeld von 4 Bit Breite: SS3= Bit[15:12], SS2=Bit[11:8], SS1=Bit[7:4], SS0=Bit[3:0]
- Einträge dort (Auswahl): 0x0 = SW-Triggered, 0x1-0x3 Analog Comparator 0-2, 0x4 Extern GPIO Triggered PB4, ..., 0xFF Selftriggered by ADC

ADC Event Multiplexer Select (ADCEMUX)



Zeitverhalten ADC und SW zeitlich (etwas) entkoppelt



```
1  // Two Channel ADC selftriggered, continuously running, FIFO empty, full and overfull
2  // Try different behavior by uncommenting a single ADC0_SAC_R value,  line 41ff.
3  // For experimental test set PE0 to GND and PE1 to Vdd
4  #include <stdint.h>
5  #include "stdio.h"
6  #include "inc/tm4c1294ncpdt.h"
7  #define MAXARRAY 20  // Block length of sampling
8
9  void main(void)
10 {
11     int i,j,k, wt; // Loop counter variables, aux. variable for waiting
12     int result1[MAXARRAY], result2[MAXARRAY]; // ADC results 2 channels
13
14     // Port and ADC Clock Gating Control
15     // Clock switch on AIN3 of ADC0  ... Pin is PE0
16     SYSCTL_RCGCGPIO_R  |= 0x00000018; // Clock Port E + D enable
17     SYSCTL_RCGCADC_R   |= 0x1; wt++;// Clock ADC0 enable  set
18
19     // Magic code for start the ADC Clocking
20     // => see tm4c1294ncpdt Datasheet, 15.3.2.7 ADC Module Clocking
21     SYSCTL_PLLFREQ0_R |= SYSCTL_PLLFREQ0_PLLPWR;     // power on the PLL
22      while(!(SYSCTL_PLLSTAT_R & SYSCTL_PLLSTAT_LOCK)); // wait till PLL has locked
23      ADC0_CC_R =  0x01 ;     wt++; // select PIOSC (internal RCOsc) as ADC analog clock
24     SYSCTL_PLLFREQ0_R &= ~SYSCTL_PLLFREQ0_PLLPWR;    // power off the PLL (s. above)
25     // end of magic code ...
26
27      GPIO_PORTE_AHB_AFSEL_R|= 0x03;     //PE0+PE1 alternative function select
28      GPIO_PORTE_AHB_AMSEL_R|= 0x03;     //PE0+PE1 analog function selecttitle
```


ADC Selftriggered Two Channels and Averaging 2 of 3

```
29        GPIO_PORTE_AHB_DEN_R   &=~0x03; // PE0+PE1 digital pin function DISABLE
30        GPIO_PORTE_AHB_DIR_R   &=~0x03; // Allow input PE0+PE1 (AIN3+AIN2)
31
32        //  for timing debug only
33        GPIO_PORTD_AHB_DEN_R      = 0x03;     // PD1 und PD0 digital enable
34        GPIO_PORTD_AHB_DIR_R      = 0x03;     // PD1 und PD0 output direction enable
35
36        // ADC init
37        ADC0_ACTSS_R    &=    0xF0;     // all sequencers off
38        ADC0_SSMUX0_R   |=0x00000032;  // Sequencer 3 channel AIN3 (PE3) and AIN2 (PE1)
39        ADC0_SSCTL0_R   |=0x00000020;  // Sequencer END1 set sequence Length= 2
40        ADC0_CTL_R=0x10;               // 3,3 V external V_ref
41        //ADC0_SAC_R=0x0;  // No averaging of samples in HW
42        //ADC0_SAC_R=0x1;  // Averaging of  2 samples in HW
43        //ADC0_SAC_R=0x2;  // Averaging over 4 samples in HW
44          ADC0_SAC_R=0x3;  // Averaging over 8 samples in HW
45        //ADC0_SAC_R=0x4;  // Averaging over 16 samples in HW
46        //ADC0_SAC_R=0x5;  // Averaging over 32 samples in HW
47        //ADC0_SAC_R=0x6;  // Averaging over 64 samples in HW
48        ADC0_EMUX_R     |=0x000F;     // continuoulsy sample enable (selftrigger seq. 0)
49
50  while(1)
51   {    ADC0_ACTSS_R    &=    0xF0;     // all sequencers off
52        printf("Measurement␣block␣starts␣now\n");
53        ADC0_ACTSS_R    |=0x01;      // sequencer 0 on
54        for (j=0;j<MAXARRAY;j++)
55        {
```


ADC Selftriggered Two Channels and Averaging 3 of 3

```
55        {
56         i=0; k=0; // aux. variable to demonstrate the waiting delay of SW for non empty FIFO
57         GPIO_PORTD_AHB_DATA_R|=0x03; // for oscilloscope two channel signal of timing only
58
59         while ((ADC0_SSFSTAT0_R & (0x100))) i++;    // wait for FIFO non empty
60         GPIO_PORTD_AHB_DATA_R&=~0x01; // for oscilloscope debug of timing only
61         result1[j] = (unsigned long) ADC0_SSFIFO0_R; // Take result out of FIFO
62         if (ADC0_SSFSTAT0_R & (1<<12)) printf ("␣FIFO␣IS␣␣FULL␣\n␣"); // only if FIFO overrun
63         while ((ADC0_SSFSTAT0_R & (0x100))) k++;    // wait for FIFO non empty
64         GPIO_PORTD_AHB_DATA_R&=~0x02; // for oscilloscope debug of timing only
65         result2[j] = (unsigned long) ADC0_SSFIFO0_R; // Take result out of FIFO
66         if (ADC0_SSFSTAT0_R & (1<<12)) printf ("␣FIFO␣IS␣␣FULL␣\n␣"); // only if FIFO overrun
67        }
68         //Test message for Full FIFO overun or not
69        if (ADC0_SSFSTAT0_R & (1<<12)) printf ("␣FIFO␣IS␣␣FULL␣\n␣");
70                          else         printf ("␣FIFO␣NOT␣FULL␣\n␣");
71           printf("Nr.␣Waiting1␣Waiting2␣Result␣Ch1␣␣Result␣Ch2␣\n");
72        for (j=0;j<MAXARRAY;j++)
73             {
74                 printf("%05d␣␣%05d␣␣␣%05d␣␣␣%4x=%04d␣␣␣%4x=%04d␣\n", j, i,k,
75                        result1[j], result1[j]* 5000UL/4096,  result2[j], result2[j]* 5000UL
       /4096  );
76             }
77        //Test for Full FIFO overun, typically must be overfull because of slow printf before
78        if (ADC0_SSFSTAT0_R & (1<<12)) printf ("␣FIFO␣IS␣␣FULL␣\n␣");
79                          else         printf ("␣FIFO␣NOT␣FULL␣\n");
80   }
81  }
```