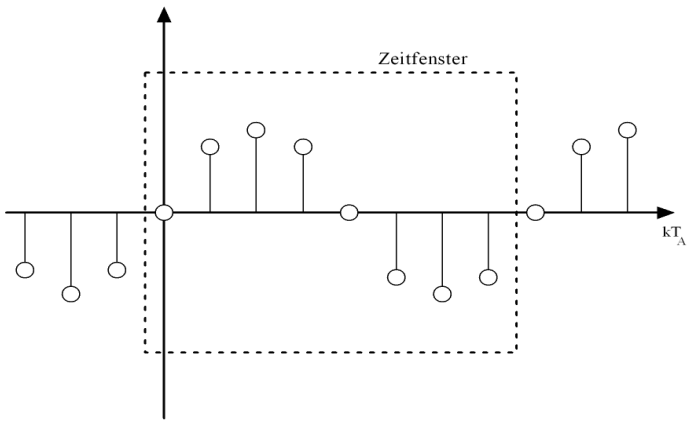
Umsetzung und Implementierung einer FFT für 8-Bit Mikrocontroller der Firma Atmel

1. Theoretischer Ansatz

1.1 Mathematischer Hintergrund



1.1 Periodisches zeitdiskretes Signal mit Abtastfenster

Das gewählte Zeitfenster mit der Periodendauer T_p umfasst die Abtastwerte n = 0 bis $(K-1)T_A$und ist somit KT_A lang. Dabei muss das Fenster für die Berechnung mindestens 2 Abtastwerte enthalten. Des Weiteren gelten folgende Zusammenhänge:

$$T_P = K \cdot T_A$$
$$T_A = \frac{T_P}{K} = \frac{1}{f_A} \qquad (1.1)$$

Der Hintergrund hierfür spiegelt sich auch im Abtasttheorem von Shannon wieder. Dieses besagt, dass man mindestens 2 Abtastwerte benötigt um ein Sinussignal rekonstruieren zu können. Dieses Sinussignal hat dann die Frequenz $f = f_A / 2$. Aufgrund der Abtastung ist allerdings das Spektrum bei $f_A / 2$ periodisch und man benötigt mindestens 4 Abtastwerte um die Grundschwingung zu errechnen.

$$f_A = \frac{K}{T_P} = K \cdot \Delta f \qquad (1.2)$$

Das heißt: Werden im Zeitbereich K Abtastwerte aufgenommen, so ergeben sich, aufgrund der Beziehung (1.2), im Frequenzbereich genau K Spektralwerte mit dem Abstand der diskreten Frequenz Δf. Die Summenformel der diskreten Fouriertransformation lautet wie folgt:

$$DFT: X(i) = \sum_{n=0}^{K-1} x(n) \cdot e^{-j2\pi n \cdot \frac{i}{K}} \qquad (1.3)$$

$$IDFT: x(n) = \frac{1}{K} \sum_{i=0}^{K-1} X(i) \cdot e^{j2\pi n \cdot \frac{i}{K}} \qquad (1.4)$$

Die Berechnung der DFT mit 8 Abtastwerten soll anhand der nachstehenden Tabelle gezeigt werden. Der Indize i entspricht der Nummerierung der diskreten Frequenz und der Indize n der Nummerierung der Abtastwerte. Laut Summenformel werden alle Spalten einer Zeile aufsummiert um die Komplexe Frequenz X(jiΔf) zu erhalten.
Als Abtastfolge wird ein Sinussignal mit $f = f_A/8$ angenommen.

$$\{x(nT_A)\} = \left\{0; \frac{1}{\sqrt{2}}; 1; \frac{1}{\sqrt{2}}; 0; -\frac{1}{\sqrt{2}}; -1; -\frac{1}{\sqrt{2}}\right\}$$

i \ n	0	1	2	3	4	5	6	7	X(jiΔf)
0	0	$\frac{1}{\sqrt{2}}$	1	$\frac{1}{\sqrt{2}}$	0	$-\frac{1}{\sqrt{2}}$	-1	$-\frac{1}{\sqrt{2}}$	0
1	0	$\frac{e^{-j\frac{\pi}{4}}}{\sqrt{2}}$	$e^{-j\frac{\pi}{2}}$	$\frac{e^{-j\frac{3\pi}{4}}}{\sqrt{2}}$	0	$\frac{e^{-j\frac{\pi}{4}}}{\sqrt{2}}$	$e^{-j\frac{\pi}{2}}$	$\frac{e^{-j\frac{3\pi}{4}}}{\sqrt{2}}$	-4j
2	0	$\frac{e^{-j\frac{\pi}{2}}}{\sqrt{2}}$	-1	$-\frac{e^{-j\frac{\pi}{2}}}{\sqrt{2}}$	0	$-\frac{e^{-j\frac{\pi}{2}}}{\sqrt{2}}$	1	$\frac{e^{-j\frac{\pi}{2}}}{\sqrt{2}}$	0
3	0	$\frac{e^{-j\frac{3\pi}{4}}}{\sqrt{2}}$	$-e^{-j\frac{\pi}{2}}$	$\frac{e^{-j\frac{\pi}{4}}}{\sqrt{2}}$	0	$\frac{e^{-j\frac{3\pi}{4}}}{\sqrt{2}}$	$-e^{-j\frac{\pi}{2}}$	$\frac{e^{-j\frac{\pi}{4}}}{\sqrt{2}}$	0
4	0	$-\frac{1}{\sqrt{2}}$	1	$-\frac{1}{\sqrt{2}}$	0	$\frac{1}{\sqrt{2}}$	-1	$-\frac{1}{\sqrt{2}}$	0
5	0	$-\frac{e^{-j\frac{\pi}{4}}}{\sqrt{2}}$	$e^{-j\frac{\pi}{2}}$	$-\frac{e^{-j\frac{3\pi}{4}}}{\sqrt{2}}$	0	$-\frac{e^{-j\frac{\pi}{4}}}{\sqrt{2}}$	$e^{-j\frac{\pi}{2}}$	$-\frac{e^{-j\frac{3\pi}{4}}}{\sqrt{2}}$	0
6	0	$-\frac{e^{-j\frac{\pi}{2}}}{\sqrt{2}}$	-1	$\frac{e^{-j\frac{\pi}{2}}}{\sqrt{2}}$	0	$\frac{e^{-j\frac{\pi}{2}}}{\sqrt{2}}$	1	$-\frac{e^{-j\frac{\pi}{2}}}{\sqrt{2}}$	0
7	0	$-\frac{e^{-j\frac{3\pi}{4}}}{\sqrt{2}}$	$-e^{-j\frac{\pi}{2}}$	$-\frac{e^{-j\frac{\pi}{4}}}{\sqrt{2}}$	0	$-\frac{e^{-j\frac{3\pi}{4}}}{\sqrt{2}}$	$-e^{-j\frac{\pi}{2}}$	$-\frac{e^{-j\frac{\pi}{4}}}{\sqrt{2}}$	4j

Tabelle 1.1.1 DFT – Berechnung einer Eingangsfolge

Betrachtet man die in Tabelle 1.1 dargestellten Multiplikationsfaktoren, so fällt auf, dass die Faktoren einer Spektrallinie X(jiΔf) sich ständig unter Änderung des Vorzeichens wiederholen. Klammert man diesen aus der Summe aus, reduziert sich die Anzahl der Multiplikationen von 8 auf 4. Diesen Effekt macht man sich bei der Berechnung der *Schnelle Fouriertransformation (FFT)* zu nutzen. Die FFT selbst stellt keine eigenständige Transformationsvorschrift dar, nur einen sehr effektiven Algorithmus zur Lösung der diskreten Fouriertransformation.

Um diese Aussage genauer Prüfen zu können, wird die gleiche Tabelle nur für den Ausdruck $e^{-j2\pi n\cdot\frac{i}{K}}$ aufgestellt.

$e^{-j2\pi n\frac{i}{K}}$		x(0)	x(T_A)	x($2T_A$)	x($3T_A$)	x($4T_A$)	x($5T_A$)	x($6T_A$)	x($7T_A$)
	i \ n	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
X(0)	**0**	1	1	1	1	1	1	1	1
X(jΔf)	**1**	1	$e^{-j\frac{\pi}{4}}$	$e^{-j\frac{\pi}{2}}$	$e^{-j\frac{3\pi}{4}}$	-1	$-e^{-j\frac{\pi}{4}}$	$-e^{-j\frac{\pi}{2}}$	$-e^{-j\frac{3\pi}{4}}$
X(j2Δf)	**2**	1	$e^{-j\frac{\pi}{2}}$	-1	$-e^{-j\frac{\pi}{2}}$	1	$e^{-j\frac{\pi}{2}}$	-1	$-e^{-j\frac{\pi}{2}}$
X(j3Δf)	**3**	1	$e^{-j\frac{3\pi}{4}}$	$-e^{-j\frac{\pi}{2}}$	$e^{-j\frac{\pi}{4}}$	-1	$-e^{-j\frac{3\pi}{4}}$	$e^{-j\frac{\pi}{2}}$	$-e^{-j\frac{\pi}{4}}$
X(j4Δf)	**4**	1	-1	1	-1	1	-1	1	-1
X(j5Δf)	**5**	1	$-e^{-j\frac{\pi}{4}}$	$e^{-j\frac{\pi}{2}}$	$-e^{-j\frac{3\pi}{4}}$	-1	$e^{-j\frac{\pi}{4}}$	$-e^{-j\frac{\pi}{2}}$	$e^{-j\frac{3\pi}{4}}$
X(j6Δf)	**6**	1	$-e^{-j\frac{\pi}{2}}$	-1	$e^{-j\frac{\pi}{2}}$	1	$-e^{-j\frac{\pi}{2}}$	-1	$e^{-j\frac{\pi}{2}}$
X(j7Δf)	**7**	1	$-e^{-j\frac{3\pi}{4}}$	$-e^{-j\frac{\pi}{2}}$	$-e^{-j\frac{\pi}{4}}$	-1	$e^{-j\frac{3\pi}{4}}$	$e^{-j\frac{\pi}{2}}$	$e^{-j\frac{\pi}{4}}$

Tabelle 1.2 DFT – Berechnung allgemein

Wie schon in Tabelle 1.1 dargestellt, treten einige Faktoren mehrfach auf. Zur weiteren Untersuchung sind die Zeilen markierte, in denen sich das Vorzeichen der Faktoren in einer gemeinsamen Zeile dreht und die Zeilen, bei denen sich das Vorzeichen nicht dreht.

Bsp.:
Zeile X(j3Δf): x(T_A) à x($5T_A$) = φ = 180°; x($2T_A$) à x($6T_A$) = φ = 180°
Zeile X(j6Δf): x(T_A) à x($5T_A$) = φ = 0° ; x($2T_A$) à x($6T_A$) = φ = 0°

Für die Spektrallinie X(jΔf) und X(j2Δf) ergeben sich folgende Summenformeln:

$$X(j\Delta f) = [x(0) - x(4T_A)]\cdot 1 + [x(1T_A) - x(5T_A)]\cdot e^{-j\frac{\pi}{4}} + [x(2T_A) - x(6T_A)]\cdot e^{-j\frac{\pi}{2}} + [x(3T_A) - x(7(T_A)]\cdot e^{-j\frac{3\pi}{4}}$$

$$X(j2\Delta f) = [x(0) + x(4T_A)]\cdot 1 + [x(1T_A) + x(5T_A)]\cdot e^{-j\frac{\pi}{2}} + [x(2T_A) + x(6T_A)]\cdot(-1) + [x(3T_A) + x(7T_A)]\cdot\left(-e^{-j\frac{\pi}{2}}\right)$$

Würde man die Untersuchung stattdessen auf die Spalten beziehen, so kommt man auf das gleiche Resultat. Es ergeben sich daher 2 Wege der Herleitung. Werden die Spalten zum Vereinfachen verwendet, so bleibt die Ausgangsreihenfolge erhalten. Praktischerweise, da alle Eingangsvariablen rein reelle Zahlen sind, wird die zweite Möglichkeit verwendet, da bei dieser die Ausgangsreihenfolge nicht sortiert werden muss. Dies ist vor allem dann wichtig, wenn der Algorithmus auf einen Mikrocontroller oder FPGA umgesetzt werden soll, da hier nur der Realteil sortiert wird.

Zusätzlich zu den Abhängigkeiten in den Zeilen, betrachtet man nun die Abhängigkeit der Spalten. Im Vergleich von Spektrallinie 2 und 6 wird deutlich, dass hier weiter vereinfacht werden kann:

$$X(j2\Delta f) = [x(0) + x(4T_A)]\cdot 1 + [x(1T_A) + x(5T_A)]\cdot e^{-j\frac{\pi}{2}} + [x(2T_A) + x(6T_A)]\cdot(-1) + [x(3T_A) + x(7T_A)]\cdot\left(-e^{-j\frac{\pi}{2}}\right)$$

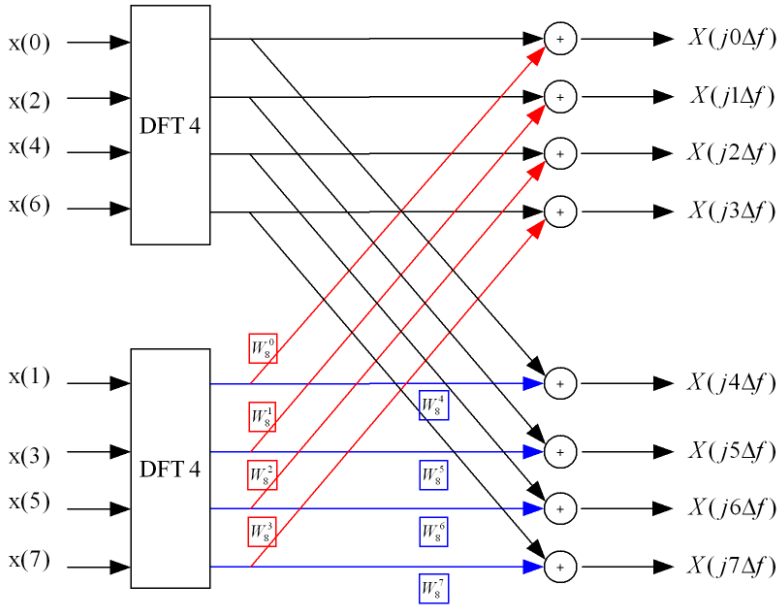
$$X(j6\Delta f) = [x(0) - x(4T_A)]\cdot 1 + [x(1T_A) + x(5T_A)]\cdot\left(-e^{-j\frac{\pi}{2}}\right) + [x(2T_A) + x(6T_A)]\cdot(-1) + [x(3T_A) + x(7T_A)]\cdot\left(e^{-j\frac{\pi}{2}}\right)$$

1.2 FFT nach dem RADIX-2 – Verfahren

Wie die Herleitung aus 1.1 ergab, können die Berechnungen von geradem und ungeradem Index vereinfacht werden. Dieses Verfahren wird im RADIX-2 angewendet, in dem die gewünschte DFT in 2 halbe DFT's für gerade und ungerade Indizes getrennt wird. Die entstandene halbe DFT kann anschließend nach dem gleichen Prinzip wieder geteilt werden, bis letztendlich nur noch 2 Werte vorhanden sind. Hierfür ist die zwingende Voraussetzung, dass die Anzahl der Abtastwerte einer Potenz von 2 entspricht.
Für die bessere Übersicht wird der so genannte Twiddle-Faktor definiert:

$$e^{-j2\pi n\frac{i}{K}} = W_K^{n\cdot i} \qquad (1.5)$$

Wird die DFT bis auf 2 Werte heruntergebrochen, so ergibt sich für den Index der Abtastwerte n = 0;1. Der Twiddle-Faktor des Abtastwertes n = 0 wird dabei automatisch zu 1. Am Beispiel der DFT-8 sieht Signalfluss der ersten Aufteilung wie folgt aus:



1.2 DFT/2 Aufteilung nach geraden und ungeraden Indizes

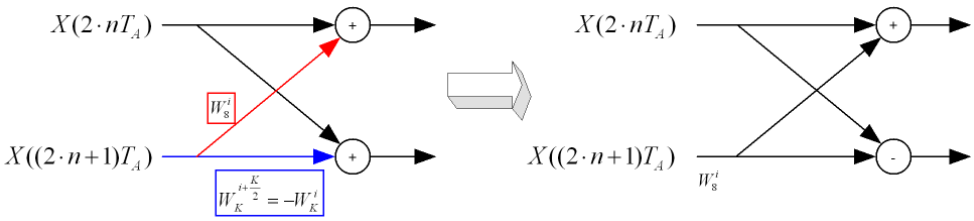
Mit diesem Hintergrundwissen, kann man nun recht einfach die Summenformel für gerade und ungerade Indizes aufstellen:

$$X(ji\Delta f) = \sum_{n=0}^{K-1} x(nT_A)\cdot e^{-j2\pi n\cdot\frac{i}{K}}$$

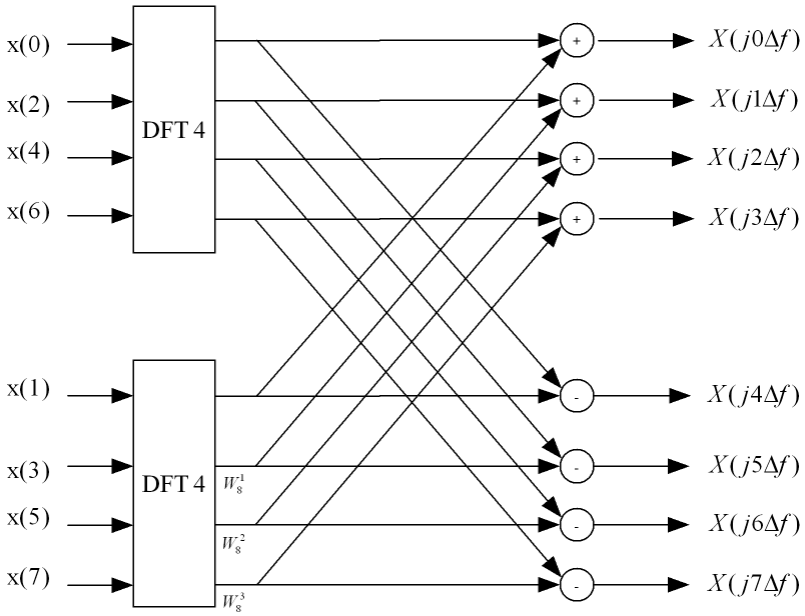
$$X(ji\Delta f) = \sum_{n=0}^{\frac{K}{2}-1} x(2\cdot nT_A)\cdot e^{-j2\pi\cdot 2n\cdot\frac{i}{K}} + \sum_{n=0}^{\frac{K}{2}-1} x((2\cdot n+1)T_A)\cdot e^{-j2\pi\cdot(2n+1)\cdot\frac{i}{K}}$$

$$X(ji\Delta f) = \sum_{n=0}^{\frac{K}{2}-1} x(2\cdot nT_A)\cdot e^{-j2\pi\cdot n\cdot\frac{i}{K/2}} + e^{-j2\pi\frac{i}{K}}\cdot\left[\sum_{n=0}^{\frac{K}{2}-1} x((2\cdot n+1)T_A)\cdot e^{-j2\pi\cdot n\cdot\frac{i}{K/2}}\right]$$

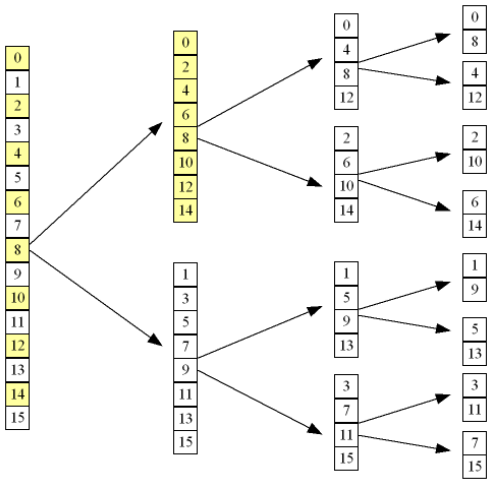
Wie schon in Kapitel 1.1 geschildert, zeigen nicht nur die Spalten Abhängigkeiten, sondern auch die Zeilen. Dies wird nun auch bei der Aufteilung im Signalgraph berücksichtigt. Jedes Pärchen kann dabei wie nachstehend gezeigt vereinfacht werden.



Der resultierende Graph sieht anschließend wie folgt aus:

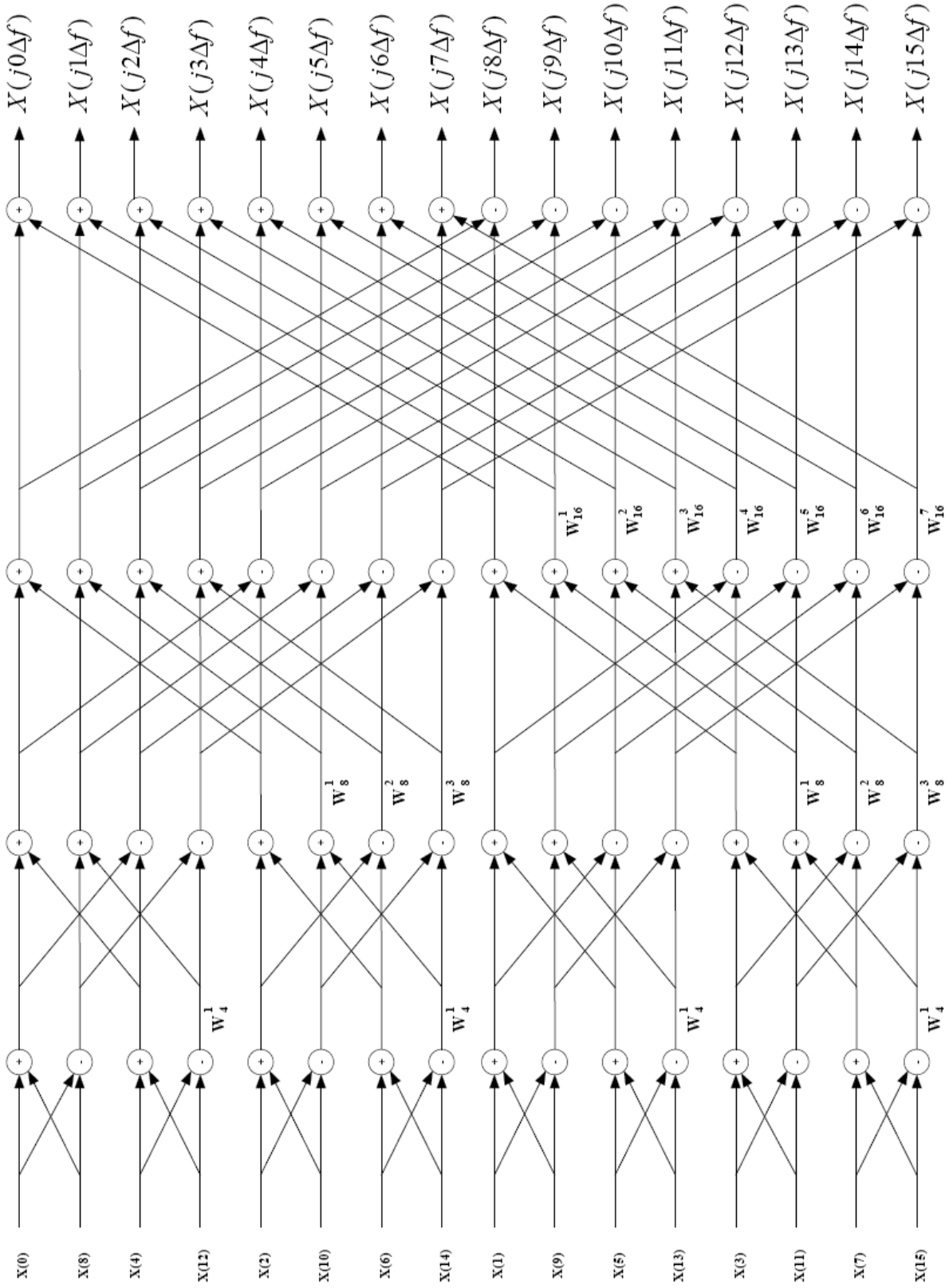


An dieser Stelle müsste nun wieder jede der beiden DFT 4 nach diesem Prinzip aufgeteilt werden. In jeder einzelnen Stufe ergeben sich daher eine unterschiedliche Anzahl von zu verarbeitenden Abtastwerten und somit andere Twiddle-Faktoren. Da das Betragsspektrum allerdings periodisch mit $f_a/2$ ist, muss die letzte Subtraktion für X(j4Δf) bis X(j7Δf) nicht berechnet werden. Für die Entwicklung eines Algorithmus für einen entsprechenden Mikrocontroller wird mit einer FFT-16 begonnen, da hier 8 Spektrallinien entstehen und sich die Berechnung noch in Grenzen hält. Eine spätere Erweiterung auf höherwertige FFT's ist kein Problem. Nachstehend dargestellt ist die maximale Aufteilung der Abtastwerte bis zu einer DFT-2, durch deren Namen dieses Berechnungsverfahren RADIX-2 gibt.



1.3 Sortierung der Abtastwerte

Butterfly – Graph für FFT 16



Bei genauer Betrachtung des Butterfly – Graphen sind einige Abhängigkeiten erkennbar, die eine Berechnung höherwertiger FFT's ermöglicht ohne für diese den Graphen aufstellen zu müssen.

Die 0.Stufe muss separat betrachtet werden. Alle nachfolgenden Stufen, mit dem Stufenindex S ergeben sich nach folgenden Regeln:

Offset der Additionspärchen: 2^S

Offset der Twiddlefaktorposition: $2^S + 1$

Twiddlefaktor: $W_{2^{S+1}}^{n} \; für \quad 1 \le n \le 2^S - 1$

1.3 Fensterung

Grundsätzlich lässt sich die Eingangsfolge direkt wie in den vorherigen Kapiteln berechnen. Dabei wird allerdings immer angenommen, dass sich ein ganzes Vielfaches der Signalfrequenz innerhalb der Abtastzeit befindet. Ist die Signalfrequenz kein ganzes Vielfaches, so entstehen beim aneinanderreihen der Abtastfenster Unstetigkeiten. Durch diese entstehen neue Spektrallinien und alte werden Aufgeweitet. Um dies zu unterdrücken werden die Abtastwerte verschieden Gewichtet. Die Wichtung kann dabei durch verschiedene Fenster erfolgen. Die Auswahl einer geeigneten Fensterfunktion ist immer ein Kompromiss zwischen Seitenbandunterdrückung und Breite der Spektrallinien. Werden die Abtastwerte nicht verändert, so entspricht dies eines Rechteckfensters. Nachstehend sind die Berechnungsvorschriften verschiedener Fensterfunktionen aufgelistet.

Bartlett-Fenster

$$\{w(k)\} = \begin{cases} \dfrac{2k}{K-1} \; für & 0 \le k \le \dfrac{K-1}{2} \\ 2 - \dfrac{2k}{K-1} \; für & \dfrac{K-1}{2} < k \le K-1 \\ 0 & sonst \end{cases}$$

Hanning-Fenster

$$\{w(k)\} = \begin{cases} 0{,}5 - 0{,}5\cos\left(\dfrac{2\pi k}{K-1}\right) für & 0 \le k \le K-1 \\ 0 & sonst \end{cases}$$

Hamming-Fenster

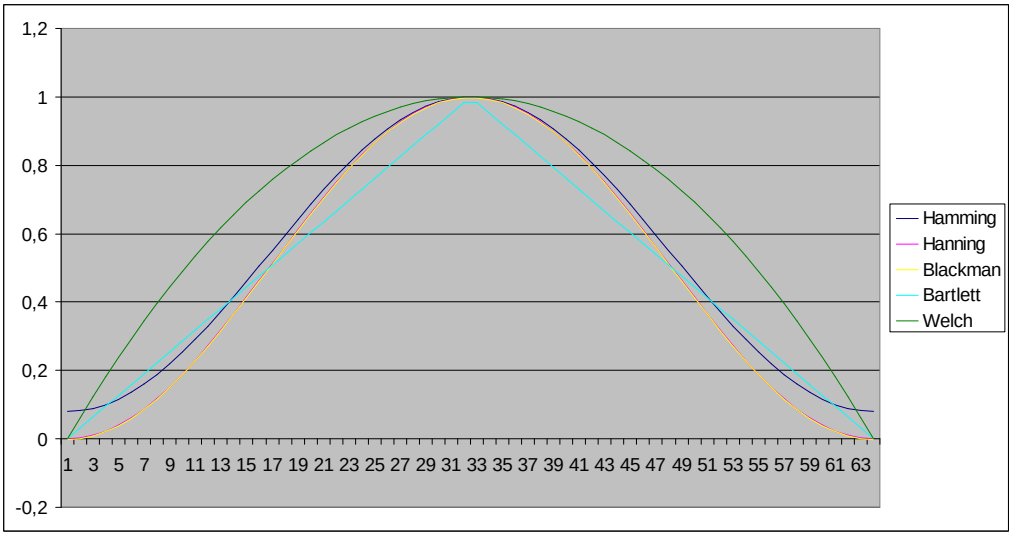
$$\{w(k)\} \begin{cases} 0{,}54 - 0{,}46\cos\left(\dfrac{2\pi k}{K-1}\right) für & 0 \le k \le K-1 \\ 0 & sonst \end{cases}$$

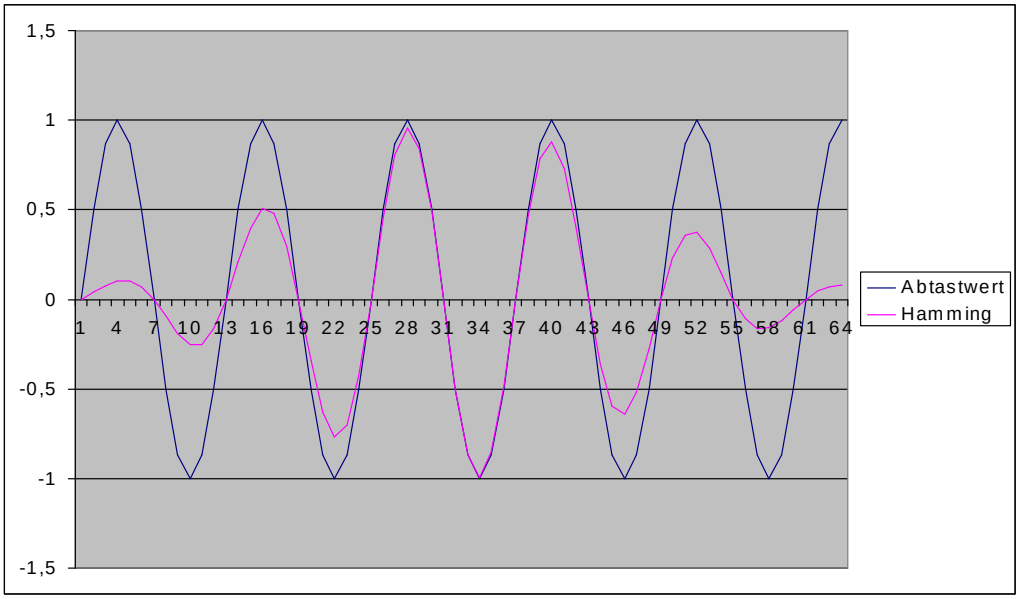
Blackman-Fenster

$$\{w(k)\} = \begin{cases} 0{,}42 - 0{,}5\cos\left(\frac{2\pi k}{K-1}\right) + 0{,}08\cos\left(\frac{4\pi}{K-1}\right) & \text{für} \quad 0 \le k \le K-1 \\ 0 & \text{sonst} \end{cases}$$

Welch-Fenster

$$\{w(k)\} = \begin{cases} 1 - \left[\frac{2k-K}{K}\right]^2 & \text{für} \quad 0 \le k \le K-1 \\ 0 & \text{sonst} \end{cases}$$





Wie man erkennen kann, würde bei der nächsten Abtastung ein Phasensprung entstehen, der neue Frequenzen erzeugt. Dieser wird durch das Hammingfenster gut unterdrückt.

2. Umsetzung

2.1 Benötigte Register und Speicher

Für die Berechnung werden immer jeweils 2 Samplespeicher benötigt. Dabei stehen nach jeder Stufe abwechselnd die Ergebnisse in dem Speicher. Des Weiteren werden folgende Register verwendet:

Register	Bezeichnung	Verwendung
R0	A_SamplelH	High - Adresse Samplespeicher A (Imaginärteil)
R1	A_SamplelL	Low - Adresse Samplespeicher A (Imaginärteil)
R2	A_SampleRH	High - Adresse Samplespeicher A (Realteil)
R3	A_SampleRL	Low - Adresse Samplespeicher A (Realteil)
R4	B_SamplelH	High -Adresse Samplespeicher B (Imaginärteil)
R5	B_SamplelL	Low -Adresse Samplespeicher B (Imaginärteil)
R6	B_SampleRH	High - Adresse Samplespeicher B (Realteil)
R7	B_SampleRL	Low - Adresse Samplespeicher B (Realteil)
R16	D0	Datenregister 0
R16	mc8s	Multiplikand für 16Bit Multiplikationsroutine
R17	D1	Datenregister 1
R17	mp8s	Multiplikator & Low-Ergebnis für 16Bit Multiplikationsroutine
R18	D2	Datenregister 2
R18	m8sH	High - Ergebnis für 16Bit Multiplikationsroutine
R19	D3	Datenregister 3
R19	mcnt8s	Schleifenzähler für 16Bit Multiplikationsroutine
R20	counter1	Zähler 1
R21	counter 2	Zähler 2

Für den ersten Codeentwurf wird für jede Stufe das entsprechende Teilprogramm aufgestellt. Später bei der Optimierung können diese aufgrund der Abhängigkeiten in gemeinsame Schleifen eingebettet werden. Der erste Schritt besteht nun in der Vorbereitung der Eingangswerte.

2.2 Eingangswerte

Für die Abtastung soll der im Mikrocontroller integrierte ADC verwendet werden. Da dieser nur unipolare Signale von 0 bis 2,5V abtasten kann, müssen alle Eingangssignale einem definierten Offset von 1,25V auferlegt bekommen. Die Amplitude des Eingangssignals darf folglich nicht größer ±1,25V betragen. Die nun abgetasteten Werte müssen, da sie ja künstlich um einen Wert von 1,25V angehoben wurden, mit Vorzeichen behandelt werden. Das heißt, alle Samplewerte werden in das 2er – Komplement übertragen, wobei der Wert 0x80 (1,25V) der Nulllinie entspricht.

Als nächsten Schritt wird nach der Abtastung von K 8Bit-Samples die gewünschte Wichtung durch die Fensterfunktion vorgenommen. Hierbei tritt nun das erste Problem auf. Alle Faktoren liegen in einem Wertebereich von {0…1}. Da keine Gleitkommafunktionen zur Verfügung stehen, müssen alle Werte entsprechend Skaliert werden. Sinnvollerweise nimmt man hier Faktoren die einer 2er – Potenz entsprechen, da dies die Reskalierung erheblich vereinfacht.

Beispiel:

k	Hamming	skalierung 128	Rundung	Fehler
0	0,08	10,16	10	0,00125
1	0,119769089	15,21067436	15	0,00164589
2	0,232199921	29,48938998	29	0,00382336
3	0,397852183	50,52722719	51	-0,00369354
4	0,588083093	74,68655282	75	-0,00244881
5	0,77	97,79	98	-0,00164063
6	0,912147817	115,8427728	116	-0,00122834
7	0,989947896	125,7233828	126	-0,00216107
8	0,989947896	125,7233828	126	-0,00216107
9	0,912147817	115,8427728	116	-0,00122834
10	0,77	97,79	98	-0,00164062
11	0,588083093	74,68655282	75	-0,00244881
12	0,397852183	50,52722719	51	-0,00369354
13	0,232199921	29,48938998	29	0,00382336
14	0,119769089	15,21067436	15	0,00164589
15	0,08	10,16	10	0,00125

Da der Mikrocontroller nur 8Bit breite Register hat, kann maximal mit einem Faktor von 256 skaliert werden. Da allerdings alle Werte im 2er-Komplement angegeben sind, bleibt als größtmöglicher Skalierungsfaktor 128 übrig. Die Multiplikation zweier 8Bit – Zahlen ergibt immer eine 16Bit-Zahl. Das Ergebnis wird anschließend durch den Skalierungsfaktor geteilt. Eine Rundung durch den Nachkommanteil sollte in späteren Programmen berücksichtigt werden.

$$10 \cdot 0{,}707 = 7{,}07$$

$$10 \cdot (0{,}707 \cdot 128) = \frac{904{,}96}{128} = 7{,}07$$

Der Vorteil bei der Verwendung eines Skalierungsfaktors der Potenz 2 ist die Division. Bei binären Zahlen bedeutet eine Verschiebung aller Bits um eine Stelle nach rechts, eine Division von 2. Folglich muss bei einer Division von 128 nur sieben mal logisch rechts geschoben werden. Dadurch kann der Rechenaufwand sehr gering gehalten werden. Das Ergebnis ist wieder eine 8Bit-Zahl, die nun für die Analyse zur

Verfügung steht. Für eine spätere Auswertung des Nachkommanteils müsste jedes aus dem Register geschobene Bit in ein weiteres Register geschoben werden. Dieses enthält dann den kompletten Nachkommaanteil. Dies wird für das erste lauffähige Programm noch nicht berücksichtigt.

Das Programm wird zu Beginn mit allen Registerdefinitionen, dem Laden des Stackpointers und der Einbindung der 16Bit-Multiplikationsroutine initialisiert. Die 16Bit – Routine entstammt von Atmel aus dem File „avr200.asm“, welches von der Firmenhomepage heruntergeladen werden kann. Aus diesem wird nur die Routine *8x8 Bit Signed Multiplication* benötigt. Da das Ergebnisregister High bereits durch einen Wert belegt sein kann, muss dieses auf dem Stack gesichert werden. Nachstehend dargestellt ist die Subroutine, die am Programmende eingefügt wird.

```
mpy8s:
push    D3          ;backup D3
sub     m8sH,m8sH   ;clear result High byte and carry
ldi     mcnt8s,8    ;init loop counter
m8s_1:
brcc    m8s_2       ;if carry (previous bit) set
add     m8sH,mc8s   ;add multiplicand to result High byte
m8s_2:
sbrc    mp8s,0      ;if current bit set
sub     m8sH,mc8s   ; subtract multiplicand from result High
asr     m8sH        ;shift right result High byte
ror     m8sL        ;shift right result L byte and multiplier
dec     mcnt8s      ;decrement loop counter
brne    m8s_1       ;if not done, loop more
pop     D3          ;restore D3
ret
```

Initialisierung:

.include "M16def.inc"

```
;---------------------------------------Vereinbarungen-------------
;
.def  A_SampleIH=r0     ;Adresse Samplespeicher A(Imaginärteil)
.def  A_SampleIL=r1
.def  A_SampleRH=r2     ;Adresse Samplespeicher A(Realteil)
.def  A_SampleRL=r3
.def  B_SampleIH=r4     ;Adresse Samplespeicher B(Imaginärteil)
.def  B_SampleIL=r5
.def  B_SampleRH=r6     ;Adresse Samplespeicher B(Realteil)
.def  B_SampleRL=r7

.def  D0=r16            ;multiplicand
.def  D1=r17            ;multiplier/result low byte
.def  D2=r18            ;result High byte
.def  D3=r19            ;loop counter
.def  counter1=r20
.def  counter2=r21
```

```
;***** Subroutine Register Variables

.def  mc8s=r16          ;multiplicand
.def  mp8s=r17          ;multiplier
.def  m8sL=r17          ;result Low byte
.def  m8sH=r18          ;result High byte
.def  mcnt8s=r19        ;loop counter

;-----------------------------------------------------------------

.org        $0
rjmp        start


start:
//load stackpointer
ldi         D0,HIGH(RAMEND)
out         SPH,D0
ldi         D0,LOW(RAMEND)
out         SPL,D0

ldi         D0,HIGH(SRAM_START)         ;lade Adresse Samplespeicher
                                        ;A (Imaginärteil)
mov         A_SampleIH,D0
ldi         D0,LOW(SRAM_START)
mov         A_SampleIL,D0

ldi         D0,HIGH(SRAM_START)         ;lade Adresse Samplespeicher
                                        ;A (Realteil)
mov         A_SampleRH,D0
ldi         D0,LOW(SRAM_START)
ldi         D1,16                       ;Offset = 16
add         D0,D1
brcc        no_carry_A_RH
inc         A_SampleRH
no_carry_A_RH:
mov         A_SampleRL,D0

ldi         D0,HIGH(SRAM_START)         ;lade Adresse Samplespeicher
                                        ;B (Imaginärteil)
mov         B_SampleIH,D0
ldi         D0,LOW(SRAM_START)
ldi         D1,32                       ;Offset 32
add         D0,D1
brcc        no_carry_B_IH
inc         B_SampleIH
no_carry_B_IH:
mov         B_SampleIL,D0

ldi         D0,HIGH(SRAM_START)         ;lade Adresse Samplespeicher
                                        ;B (Realteil)
mov         B_SampleRH,D0
ldi         D0,LOW(SRAM_START)
ldi         D1,48                       ;Offset 48
add         D0,D1
brcc        no_carry_B_RH
inc         B_SampleRH
no_carry_B_RH:
mov         B_SampleRL,D0
```

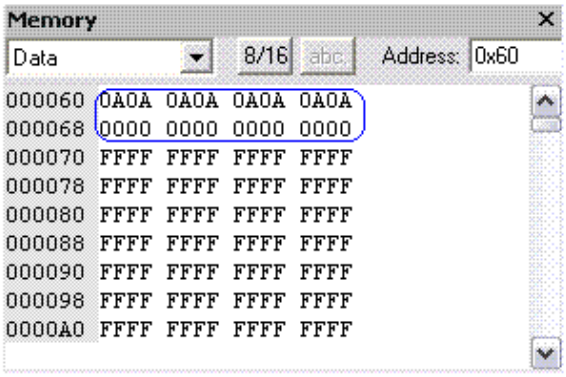
Die Adressen der einzelnen Samplespeicher im RAM sind nun geladen. Dabei liegt zu Beginn der Imaginärteil des Samplespeichers A im RAM und nach 16 Werten anschließend der Realteil des Samplespeichers A. Wird die Wertigkeit der FFT verändert, so muss das Offset angepasst werden.
Anschließend an die Definition könnte eine Benutzerauswahl des zu verwendenden Fensters durchgeführt werden. Daraufhin erfolgt der Abtastvorgang von 16 aufeinanderfolgenden Werten, der durch einen Timer ausgelöst wird. Die eingestellte Timerzeit entspricht dabei der Abtastfrequenz f_A und bestimmt somit die diskrete Frequenz Δf, also den Frequenzabstand der Spektrallinien. Dieser Vorgang soll später Umgesetzt werden. Zu Beginn werden im Simulator entsprechende Abtastwerte vorgegeben.
Diese 16 Werte werden zu Beginn in den Imaginärspeicher A geschrieben, dies ist nur ein zweckentfremdeter Zwischenspeicher da diese Anschließend in den Realteilspeicher einsortiert werden müssen. Die erste Routine ist die Fensterung. Die Werte werden dabei um den Faktor 128 Skaliert im Programm abgelegt und durch den Z-pointer geladen.

Bsp:

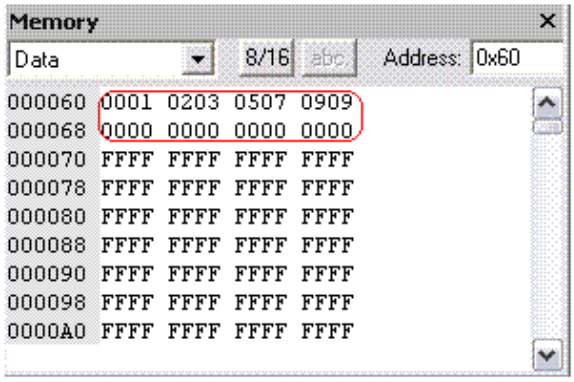
```
;************************************************************
;* Fensterfunktion
;*
;* Number of Words    : 16
;* Number of cycles   : 2070 (incl."mpy8s" routine)
;* low register used  : r0,r1
;* high register used : r16-r21
;************************************************************

ldi         ZH,HIGH(Hamming_Window<<1)    ;Pointer Fensterfunkt.laden
ldi         ZL,LOW(Hamming_Window<<1)
mov         XH,A_SampleIH                 ;Pointer ADC-Werte laden
mov         XL,A_SampleIL
ldi         counter2,16
loop_fensterung:
ld          mp8s,X                        ;lade Samplewert
lpm         mc8s,Z+                       ;lade Gewichtsfaktor
rcall       mpy8s                         ;8*8Bit = 16Bit - Routine
ldi         counter1,7
rescale_fensterung:                       ;rescale(I*x*128)/128 --> 7x
                                          ;shieben rechts
lsr         m8sH
ror         m8sL
dec         counter1
brne        rescale_fensterung
st          X+,m8sL                       ;save Samplewert
dec         counter2                      ;Alle Samples durchlaufen?
brne        loop_fensterung


Hamming_Window:
.db   10,15,29,51,75,98,116,126,126,116,98,75,51,29,15,10
Rectangle_Window:
.db   128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128
```

Hamming	Eingangsfolge	Ergebnis
10	10	0
15	10	1
29	10	2
51	10	3
75	10	5
98	10	7
116	10	9
126	10	9
126	0	0
116	0	0
98	0	0
75	0	0
51	0	0
29	0	0
15	0	0
10	0	0

Vorgabe der Eingangswerte



Ergebnis Fensterung



2.3 Bit-Reversal-Sort

Die nächste Aufgabe besteht in der Sortierung der Eingangsreihenfolge. Das Sortierverfahren soll Anhand von 8 Werten in nachstehender Tabelle erläutert werden:

Reihe	Binärwert	Augangsfolge	Binärwert
0	000	0	000
1	001	4	100
2	010	2	010
3	011	6	110
4	100	1	001
5	101	5	101
6	110	3	011
7	111	7	111

Wie man erkennen kann, so entspricht die Ausgangsfolge immer der umgekehrten Bitreihenfolge des Eingangswertes. Somit kann direkt die Ausgangfolge generiert werden, ohne jeden Zwischenschritt berechnen zu müssen. Dafür wird der Adressoffset (0…K-1) im Register genau um die Zweierpotenz (im obigen Beispiel 3)

nach rechts verschoben. Der aus dem Register geschobene Wert wird anschließend in ein weiteres Register durch rotieren links eingefügt.

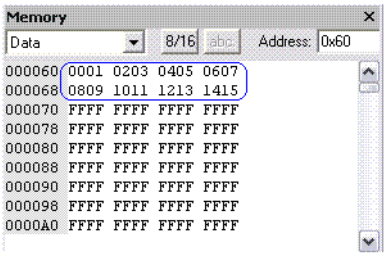
Für die 16Bit Routine bedeutet dies, dass ein Zählwert 0…15 auf die Startadresse der Abtastwerte im RAM gelegt wird. Dieser Offsetwert wird dann durch Schieben und Rotieren in die umgekehrte Bitfolge gebracht. Dieser neue Offsetwert wird auf den Start der Zieladresse aufaddiert und die Werte entsprechend von Quell nach Zielspeicher kopiert.

```
;************************************************************
;* Bit-Reversal-Sort   (Samplespeicher I --> Samplespeicher R)
;*
;* Number of Words    : 25
;* Number of cycles   : 672
;* low register used  : r0,r1,r2,r3
;* high register used : r16-r21
;************************************************************
ldi         counter1,0                          ;Anzahl Samples
sort_next:
mov         XH,A_SampleIH                       ;Adresspointer
Speicher I laden
mov         XL,A_SampleIL
mov         D0,A_SampleIL                       ;Samplespeicher IL
mov         D1,A_SampleRL                       ;Samplespeicher RL
mov         D2,counter1                         ;Offset
clr         D3
ldi         counter2,4                          ;Anzahl der Adressbits
bitshift:
ror         D2                                  ;shift right --> carry
rol         D3                                  ;shift left <-- carry
                                                ;(D3=reversal)
dec         counter2
brne        bitshift
add         XL,counter1                         ;Offset Speicher I
brcc        Sort_SpeicherI_no_Carry
inc         XH
Sort_SpeicherI_no_Carry:
ld          D2,X                                ;D2=Samplewert
mov         XH,A_SampleRH
mov         XL,A_SampleRL
add         XL,D3
brcc        Sort_SpeicherR_no_Carry
inc         XH
Sort_SpeicherR_no_Carry:
st          X,D2                                ;store Samplewert on
                                                ;new position
inc         counter1
cpi         counter1,16
brne  sort_next
```

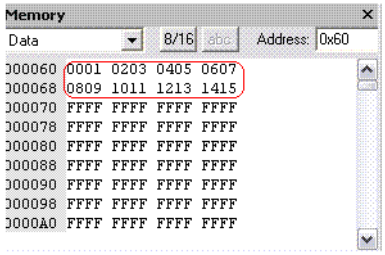
Für alle nachfolgenden Tests wird die Rechteckfensterfunktion verwendet, da man für die Sortierung verschiedene Testwerte benötigt, die dann alle nur mit Faktor 1 gewichtet werden.

Als Testfolge der Abtastwerte aufeinanderfolgende Werte verwendet. (siehe Abb.)

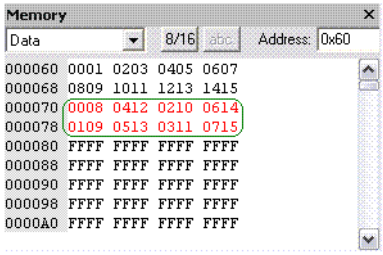
Eingangswerte



Werte nach Rechteckfensterung

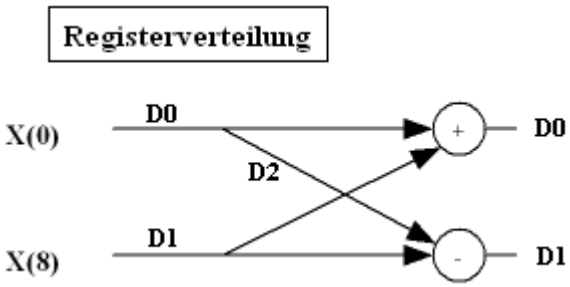


Sortierte Wert



2.4 FFT Stufe 0

In der Stufe 0 werden aufgrund des Twiddlefaktors $W_2^0 = 1$ keine Multiplikationen durchgeführt. Zudem stehen die Eingangswerte gewichtet und in richtiger Reihenfolge im Realteil des Samplespeicher A. Da in dieser Stufe der Twiddlefaktor rein reell ist, muss der Imaginärteil nicht verarbeitet werden. Dieser ist Null und muss für die weiteren Berechnungsschritte noch gelöscht werden.




```
;*********************************************************************
;* clear Imaginärteil
;*
;* Number of words    : 7
;* Number of cycles       : 83
;* used low register  : r0,r1
;* used high register : r16,r20
;*********************************************************************
ldi         counter1,16                         ;Anzahl der Samples
mov         XH,A_SampleIH                       ;load Imaginärteil A
mov         XL,A_SampleIL
clr         D0
clear_IM:
st          X+,D0                               ;setze Wert 0x00
dec         counter1
brne        clear_IM


;*********************************************************************
;* Stage 0
;*
;* Number of words   : 29
;* Number of cycles  : 281
;* used low register : r0-r7
;* used high register: D0-D2;r20,r21
;*********************************************************************
ldi         counter1,0                  ;Anzahl der Iterationen
ldi         counter2,2                  ;Offset Hilfsgröße (jede
                                        ;berechnung 2 Schritte
```

```
                                        ;weiter)
ldi         D3,0                        ;Offset-counter
Real_Add_Stage0:
mov         XH,A_SampleRH               ;load Pointer Realteil A
mov         XL,A_SampleRL
add         XL,D3                       ;add Offset
brcc        no_carry_s0_1
inc         XH
no_carry_s0_1:
ld          D0,X+                       ;D0 = Sample 0
ld          D1,X+                       ;D1 = Sample 1
mov         D2,D0                       ;Backup D0
add         D0,D1                       ;D0=D0+D1
brge        stage0_no_overflow1         ;Signed-flag clear?
sec                                     ;set carry
stage0_no_overflow1:
ror         D0                          ;D0=D0/2
sub         D2,D1                       ;D2=D2-D1
brge        stage0_no_overflow2         ;signed flag clear?
sec
stage0_no_overflow2:
ror         D2                          ;D2=D2/2
mov         XH,B_SampleRH               ;load Pointer Realteil B
mov         XL,B_SampleRL
add         XL,D3                       ;add Offset
brcc        no_carry_s0_2
inc         XH
no_carry_s0_2:
st          X+,D0                       ;store Samplespeicher B
st          X+,D2

add         D3,counter2                 ;next pair
cpi         D3,18                       ;Alle Samples verarbeitet?
brne        Real_Add_Stage0


;*****************************************************************
;* clear Imaginärteil B
;*
;* Number of words    : 7
;* Number of cycles       : 83
;* used low register  : r0,r1
;* used high register : r16,r20
;*****************************************************************
ldi         counter1,16                 ;Anzahl der Samples
mov         XH,B_SampleIH               ;load Imaginärteil B
mov         XL,B_SampleIL
clr         D0                          ;setze Wert 0x00
clear_IM2:
st          X+,D0
dec         counter1
brne        clear_IM2
```

Diese Berechnung wurde mittels Excel vorbereitet, um die Ergebnisse der einzelnen Stufen vergleichen zu können.

X[k]	Samplewert [V]	X[k] "reversal"	Samplewerte [V]	Re{A[k]}	Im{A[k]}
0	0	0	0	1,25	0
1	0	8	2,5	-1,25	0
2	0	4	0	1,25	0
3	0	12	2,5	-1,25	0
4	0	2	0	1,25	0
5	0	10	2,5	-1,25	0
6	0	6	0	1,25	0
7	0	14	2,5	-1,25	0
8	2,5	1	0	1,25	0
9	2,5	9	2,5	-1,25	0
10	2,5	5	0	1,25	0
11	2,5	13	2,5	-1,25	0
12	2,5	3	0	1,25	0
13	2,5	11	2,5	-1,25	0
14	2,5	7	0	1,25	0
15	2,5	15	2,5	-1,25	0

Ergebnisse der Simulation im 2er – Komplement: