6.6 OPTIMIZED RADIX-4 DIF FFT

This section explores changes to the radix-4 FFT program to increase its execution speed. Specifically, changes in the first and last stages, data structures and program flow are discussed.

6.6.1 First Stage Modifications

In the first stage, there are N/4 butterflies and only one group and therefore the group loop is not needed. The butterfly loop and the group loop can be combined into one. Because each loop requires several setup instructions to initialize the counter and other registers, combining the two loops enhances performance. The code for these combined loops is shown in Listing 6.28.

```
{————————— Stage 1 ————————————}



stage1:  I0=^inplace;                {in ->Xa,Xc}
         I1=^inplace+Nov2;           {in+N/2 ->Xb,Xd}
         I2=^inplace+1;              {in+1 ->Ya,Yc}
         I3=^inplace+Nov2+1;         {in+N/2+1 ->Yb,Yd}
         I5=^cos_table;
         I6=^cos_table;
         I7=^cos_table;

         M0=N;                       {N,    skip forward to dual node}
         M1=-N;                      {-N,   skip back to primary node}
         M2=-N+2;                    {-N+2, skip to next butterfly}
         M5=Nov4+1;          {N/4 + groups/stage*1, Cb Sb offset}
         M6=Nov4+2;          {N/4 + groups/stage*2, Cc Sc offset}
         M7=Nov4+3;          {N/4 + groups/stage*3, Cd Sc offset}
         AX0=DM(I0,M0);              {get first Xa}
         AY0=DM(I0,M1);              {get first Xc}
         AR=AX0-AY0, AX1=DM(I2,M0); {Xa-Xc,get first Ya}
         SR=LSHIFT AR(LO), AY1=DM(I2,M1); {SR1=Xa-Xc,get first Yc}
         CNTR=Nov4;                  {Bfly/group, stage one}
         DO stg1bfy UNTIL CE;

         <butterfly code here>

stg1bfy:

{————————— end Stage 1 ————————————}
```

Listing 6.28 First Stage with Combined Butterfly and Group Loops

6.6.2 Last Stage Modifications

In the last stage, all the twiddle factors are 1, therefore the multiplications can be removed from the butterfly. The butterfly equations reduce to the following:

$$x_a' = x_a + x_b + x_c + x_d$$

$$y_a' = y_a + y_b + y_c + y_d$$

$$x_b' = x_a - x_c + y_b - y_d$$

$$y_b' = y_a - y_c - x_b + x_d$$

$$x_c' = x_a - x_b + x_c - x_d$$

$$y_c' = y_a - y_b + y_c - y_d$$

$$x_d' = x_a - x_c - y_b + y_d$$

$$y_d' = y_a - y_c + x_b - x_d$$

These reduced equations can be computed using a simplified butterfly algorithm. This speed improvement entails a separate butterfly module for the last stage.

The general set of butterfly equations (including twiddle factor multiplications) can be implemented in 30 cycles using the code in Listing 6.29. On the other hand, the simplified equations of the last stage can be computed in 20 cycles using the code in Listing 6.30. A modest increase in program memory requirements is traded for a significant increase in the performance of this core loop.

```
DO stg1bfy UNTIL CE;
   AR=AX0+AY0, AX0=DM(I1,M0);       {AR=xa+xc, AX0=xb}
   MR0=AR, AR=AX1+AY1;              {MR0=xa+xc, AR=ya+yc}
   MR1=AR, AR=AX1-AY1;              {MR1=ya+yc, AR=ya-yc}
   SR=SR OR LSHIFT AR (HI), AY0=DM(I1,M1);
                                    {SR1=ya-yc, AY0=xd}
   AF=AX0+AY0, AX1=DM(I3,M0);       {AF=xb+xd, AX1=yb}
   AR=MR0+AF, AY1=DM(I3,M1);        {AR=xa+xb+xc+xd, AY1=yd}
   DM(I0,M0)=AR, AR=MR0-AF;
                      {output x'a=(xa+xb+xc+xd), AR=xa+xc-xb-xd}
   AF=AX1+AY1, MX0=AR;              {AF=yb+yd, MX0=xa+xc-xb-xd}
   AR=MR1+AF, MY0=DM(I6,M4);        {AR=ya+yc+yb+yd, MY0=(Cc)}
```

```
                DM(I2,M0)=AR, AR=MR1-AF;         {output y’a, AR=ya+yc-yb-yd}
                MR=MX0*MY0(SS), MY1=DM(I6,M6);
                                           {MR=(xa+xc-xb-xd)(Cc), MY1=(Sc)}
                MR=MR+AR*MY1(RND);         {MR=(xa-xb+xc-xd)(Cc)+(ya-yb+yc-yd)(Sc)}
                DM(I0,M2)=MR1, MR=AR*MY0(SS);
                               {output x’c=xa-xb+xc-xd)(Cc)+(ya-yb+yc-yd)(Sc)}
                                                 {MR=(ya+yc-yb-yd)(Cc)}
                MR=MR-MX0*MY1(RND), MY0=DM(I5,M4);
                         {MR=(ya+yc-yb-yd)(Cc)-(xa+xc-xb-xd)(Sc), MY0=(Cb)}
                DM(I2,M2)=MR1, AR=AX0-AY0;
                         {output y’c=(ya+yc-yb-yd)(Cc)-(xa+xc-xb-xd)(Sc)}
                                                 {AR=xb-xd}
                AY0=AR, AF=AX1-AY1;              {AY0=xb-xd, AF=yb-yd}
                AR=SR0-AF, MY1=DM(I5,M5);        {AR=xa-xc-(yb-yd), MY1=(Sb)}
                MX0=AR, AR=SR0+AF;               {MX0=xa-xc-yb+yd, AR=xa-xc+yb-yd}
                SR0=AR, AR=SR1+AY0;        {SR0=xa-xc+yb-yd, AR=ya-yc+xb-xd}
                MX1=AR, AR=SR1-AY0;        {MX1=ya-yc+xb-xd, AR=ya-yc-(xb-xd)}
                MR=SR0*MY0(SS), AX0=DM(I0,M0);
                               {MR=(xa-xc+yb-yd)(Cb), AX0=xa of next bfly}
                MR=MR+AR*MY1(RND), AY0=DM(I0,M1);
                               {MR=(xa-xc+yb-yd)(Cb)+(ya-yc-xb+xd)(Sb)}
                                                 {AY0=xc of next bfly}
                DM(I1,M0)=MR1, MR=AR*MY0(SS);
                         {output x’b=(xa-xc+yb-yd)(Cb)+(ya-yc-xb+xd)(Sb)}
                                                 {MR=ya-yc-xb+xd)(Cb)}
                MR=MR-SR0*MY1(RND), MY0=DM(I7,M4);
                               {MR=(ya-yc-xb+xd)(Cb)-(xa-xc+yb-yd)(Sb)}
                                                 {MY0=(Cd)}
                DM(I3,M0)=MR1, MR=MX0*MY0(SS);
                         {output y’b=(ya-yc-xb+xd)(Cb)-(xa-xc+yb-yd)(Sb)}
                                                 {MR=(xa-yb-xc+yd)(Cd)}
                MY1=DM(I7,M7), AR=AX0-AY0;       {MY1=(Sd), AR=xa-xc}
                MR=MR+MX1*MY1(RND), AX1=DM(I2,M0);
                               {MR=(xa-yb-xc+yd)(Cd)+(ya+xb-yc-xd)(Sd)}
                                                 {AX1=ya of next bfly}
                DM(I1,M2)=MR1, MR=MX1*MY0(SS);
                         {output x’d=(xa-yb-xc+yd)(Cd)+(ya+xb-yc-xd)(Sd)}
                                                 {MR=(ya+yb-yc-yd)(Cd)}
                MR=MR-MX0*MY1(RND), AY1=DM(I2,M1);
                                     {MR=(ya+yb-yc-yd)(Cd)-(xa-xc-yb+yd)Sd}
                                                 {yc of next bfly}
stg1bfy:        DM(I3,M2)=MR1, SR=LSHIFT AR(LO);
                               {output y’d=(ya+xb-yc-xd)Cd-(xa-xc-yb+yd)Sd}
                                                 {SR0=ya-yc of next bfly}
```

Listing 6.29 Unmodified Butterfly

```
                DO laststgbfy UNTIL CE;
                   AR=AX0-AY0, AX1=DM(I2,M0);       {AR=xa-xc, AX1=ya}
                   SR=LSHIFT AR(LO), AY1=DM(I2,M1); {SR0=xa-xc, AY1=yc}
                   AR=AX0+AY0, AX0=DM(I1,M0);       {AR=xa+xc, AX0=xb}
                   MR0=AR, AR=AX1+AY1;              {MR0=xa+xc, AR=ya+yc}
                   MR1=AR, AR=AX1-AY1;              {MR1=ya+yc, AR=ya-yc}
                   SR=SR OR LSHIFT AR (HI), AY0=DM(I1,M1); {SR1=ya-yc, AY0 xd}
                   AF=AX0+AY0, AX1=DM(I3,M0);       {AF=xb+xd, AX1=yb}
                   AR=MR0+AF, AY1=DM(I3,M1);        {AR=xa+xc+xb+xd, AY1=yd}
                   DM(I0,M0)=AR, AR=MR0-AF;
                                  {output x'a=xa+xc+xb+xd, AR=xa+xc-(xb+xd)}
                   DM(I0,M0)=AR, AF=AX1+AY1; {output x'c=xa+xc-(xb+xd), AF=yb+yd}
                   AR=MR1+AF;                       {AR=ya+yb+yc+yd}
                   DM(I2,M0)=AR, AR=MR1-AF;
                                  {output y'a=ya+yc+yb+yd, AR=ya+yc-(yb+yd)}
                   DM(I2,M0)=AR, AR=AX0-AY0; {output y'c=ya+yc-(yb+yd), {AR=xb-xd}
                   AX0=DM(I0,M0);                   {AX0=xa of next group}
                   AF=AX1-AY1, AY1=AR;              {AF=yb-yd, AY1=xb-xd}
                   AR=SR0+AF, AY0=DM(I0,M1);
                                        {AR=xa-xc+yb-yd, AY0=xc of next group}
                   DM(I1,M0)=AR, AR=SR0-AF;
                                        {output x'b=xa-xc+yb-yd, AR=xa-xc-(yb-yd)}
                   DM(I1,M0)=AR, AR=SR1-AY1;
                                  {output x'd=xa-xc-(yb-yd), AR=ya-yc+(xb-xd)}
                   DM(I3,M0)=AR, AR=SR1+AY1;
                                  {output y'b=ya-yc+(xb-xd), AR=ya-yc-(xb-xd)}
laststgbfy:        DM(I3,M0)=AR;                    {output y'd=ya-yc-(xb-xd)}
```

Listing 6.30 Simplified Last Stage Butterfly

In addition to having its own butterfly algorithm, the last stage has only one butterfly in each of its N/4 groups, so the group and butterfly loops can be combined into one, reducing the loop setup overhead.

6.6.3 Program Structure Modifications

While the nested loop structure is efficient in terms of program storage requirements, it does not take advantage of the unique properties of the first and last stages as outlined above. In implementing the modifications to these stages it becomes convenient to restructure the algorithm. The outermost loop (stage loop) can be removed and the stage setup instructions can be done sequentially, in a nonrecursive manner. Each stage can have its own setup instructions followed by a call to a subroutine that executes the remaining two nested loops (group and butterfly loops). The program with the modified structure is shown in Listing 6.31.

```
{———————————— Stage 1 ————————————}

stage1:     I0=^inplace;                            {in ->Xa,Xc}
            I1=^inplace+Nov2;                       {in+N/2 ->Xb,Xd}
            I2=^inplace+1;                          {in+1 ->Ya,Yc}
            I3=^inplace+Nov2+1;                     {in+N/2+1 ->Yb,Yd}
            I5=^cos_table;
            I6=^cos_table;
            I7=^cos_table;

            M0=N;                             {N,    skip forward to dual node}
            M1=-N;                            {-N,   skip back to primary node}
            M2=-N+2;                          {-N+2, skip to next butterfly}
            M5=Nov4+1;                        {N/4+groups/stage*1, Cb Sb offset}
            M6=Nov4+2;                        {N/4+groups/stage*2, Cc Sc offset}
            M7=Nov4+3;                        {N/4+groups/stage*3, Cd Sc offset}
            AX0=DM(I0,M0);                    {get first Xa}
            AY0=DM(I0,M1);                    {get first Xc}
            AR=AX0-AY0, AX1=DM(I2,M0);        {Xa-Xc,get first Ya}
            SR=LSHIFT AR(LO), AY1=DM(I2,M1);        {SR1=Xa-Xc,get first Yc}
            CNTR=Nov4;                        {Bfly/group, stage one}
            DO stg1bfy UNTIL CE;
```

< butterfly code here >

```
stg1bfy:


{———————————— Stage 2 ————————————}

stage2:     I0=^inplace;            {in -> Xa,Xc}
            I1=^inplace+128;        {in+N/8 -> Xb,Xd}
            I2=^inplace+1;          {in+1 -> Ya,Yc}
            I3=^inplace+129;        {in+N/8+1 -> Yb,Yd}
            M0=Nov4;                {N/4, skip forward to dual node}
            M1=-Nov4;               {-N/4, skip back to primary node}
            M2=-Nov4+2;             {-N/4+2, skip to next butterfly}
            M3=384;                 {N*3/8,  skip to next group}
            M5=Nov4+4;              {N/4+groups/stage*1, Cb Sb offset}
            M6=Nov4+8;              {N/4+groups/stage*2, Cc Sc offset}
            M7=Nov4+12;             {N/4+groups/stage*3, Cd Sd offset}
            SI=64;                  {Bfly/group, save counter for inner loop}
            DM(bfy_count)=SI;
            CNTR=4;                 {groups/stage}
            CALL mid_stg;           {do stage 2}
```

(listing continues on next page)

```
{————————————— Stage 3 —————————————}

stage3:     I0=^inplace;            {in -> Xa,Xc}
            I1=^inplace+32;         {in+N/32 -> Xb,Xd}
            I2=^inplace+1;          {in+1 -> Ya,Yc}
            I3=^inplace+33;         {in+N/32+1 -> Yb,Yd}
            M0=64;                  {N/16, skip forward to dual node}
            M1=-64;                 {-N/16, skip back to primary node}
            M2=-62;                 {-N/16+2, skip to next butterfly}
            M3=96;                  {N*3/32, skip to next group}
            M5=Nov4+16;             {N/4+groups/stage*1, Cb Sb offset}
            M6=Nov4+32;             {N/4+groups/stage*2, Cc Sc offset}
            M7=Nov4+48;             {N/4+groups/stage*3, Cd Sd offset}
            SI=16;                  {Bfly/group, save counter for inner loop}
            DM(bfy_count)=SI;
            CNTR=16;                {groups/stage}
            CALL mid_stg;           {do stage 3}

{————————————— Stage 4 —————————————}

stage4:     I0=^inplace;            {in -> Xa,Xc}
            I1=^inplace+8;          {in+N/128 -> Xb,Xd}
            I2=^inplace+1;          {in+1 -> Ya,Yc}
            I3=^inplace+9;          {in+N/128+1 -> Yb,Yd}
            M0=16;                  {N/64, skip forward to dual node}
            M1=-16;                 {-N/64, skip back to primary node}
            M2=-14;                 {-N/64+2, skip to next butterfly}
            M3=24;                  {N*3/128, skip to next group}
            M5=Nov4+64;             {N/4 +groups/stage*1, Cb Sb offset}
            M6=Nov4+128;            {N/4 +groups/stage*2, Cc Sc offset}
            M7=Nov4+192;            {N/4 +groups/stage*3, Cd Sd offset}
            SI=4;                   {Bfly/group, save counter inner loop}
            DM(bfy_count)=SI;
            CNTR=64;                {groups/stage}
            CALL mid_stg;           {do stage 4}


{——————— Last Stage (No Multiplies) ————————}


laststage:  I0=^inplace;            {in ->Xa,Xc}
            I1=^inplace+2;          {in+N/512 ->Xb,Xd}
            I2=^inplace+1;          {in+1 ->Ya,Yc}
            I3=^inplace+3;          {in+N/512+1 ->Yb,Yd}
            M0=4;                   {N/256, skip forward to dual node}
            M1=-4;                  {-N/256, skip back to primary node}
            AX0=DM(I0,M0);          {first Xa}
            AY0=DM(I0,M1);          {first Xc}
            CNTR=Nov4;              {groups/stage}
            DO laststgbfy UNTIL CE;

                < last stage butterfly code here >

laststgbfy:
```

Listing 6.31 Modified Program Structure

Program size is increased in exchange for clarity and flexibility. The nonrecursive structure allows future modifications to be incorporated more readily than does the 3-nested-loop structure, because the changes can be applied to a particular stage without affecting the other stages.

6.6.4 Data Structure Modifications

6.6.4.1 Cosine Table

In the unoptimized FFT, real and imaginary parts of the twiddle factors are stored separately in two array structures, each of length N. In the optimized FFT, a single array of size N is used to store the real values (cosine values). Using an addressing modify value of –N/4 (–π/2 in terms of angular displacement), the imaginary values (sine values) can be derived from the cosine table. This is based on the trigonometric identity

$$\sin(x) = \cos(x-\pi/2)$$

This modification results in a 50% improvement in program data memory requirements.

For example, $W_{1024}^{256} = \cos[256] - j\sin[256]$
$= \cos[256] - j\cos[0]$

where cos[x] is defined as cos (2πx/N). The structure of the modified table is shown in Figure 6.13, on the next page.

6.6.4.2 In-Place Array

The computations of the FFT are performed in place, that is, the output data occupy the same memory locations as the input data (the *inplace* array). This approach simplifies indexing and reduces the amount of memory required. Input samples are stored in the 2N array *inplace*; the

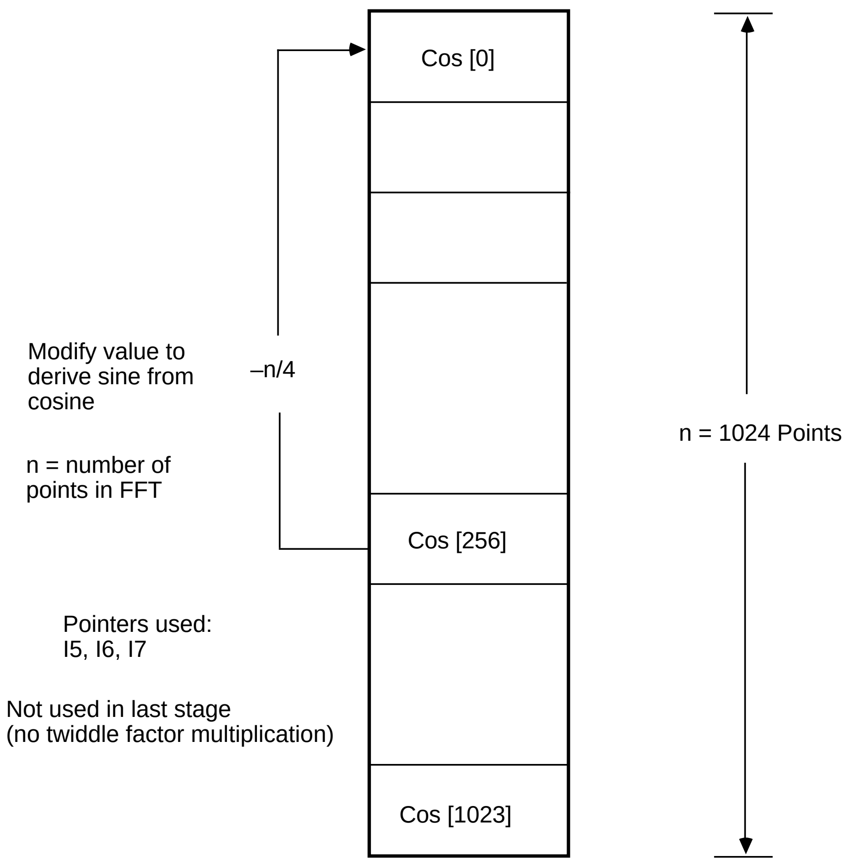
Cos [0]
Modify value to derive sine from cosine
–n/4
n = number of points in FFT
Cos [256]
Pointers used: I5, I6, I7
Not used in last stage (no twiddle factor multiplication)
Cos [1023]
n = 1024 Points

Figure 6.13 Cosine Table for 1024-Point FFT

real and imaginary values are interleaved, as shown in Figure 6.14. The program uses pointers I0, I1, I2 and I3 to read and write this array, except in the modified last stage, where I4, I5, I6 and I7 are used also (all pointers are used in last stage).

6.6.5 Digit-Reversing

In an in-place DIF FFT computation, the output is not in sequential order. Repeatedly subdividing the input data sequences produces a scrambled (nonsequentially-ordered) output. For a radix-4 FFT, the output is in digit-reversed order.

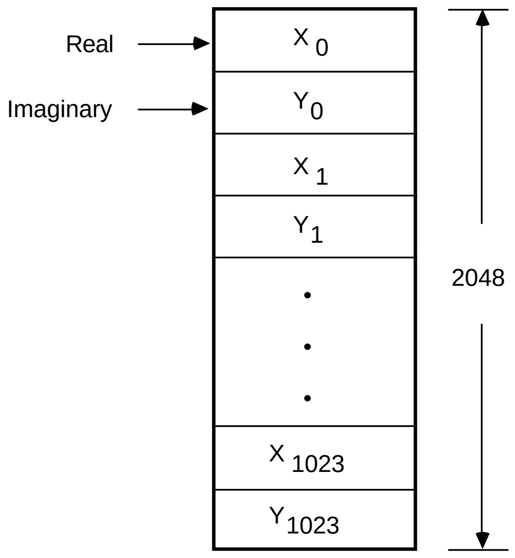

Figure 6.14 Interleaved Structure

Two methods to unscramble the digit-reversed output into sequential order are outlined below:

- A separate routine unscrambles the FFT output. This routine reads the digit-reversed output and writes it back in sequential order. It has the disadvantage of adding to the total execution time of the program.

- In a radix-4 algorithm, the butterfly can be modified to to produce a bit-reversed FFT output instead of a digit-reversed output. Combining this with the built-in bit-reverse mode of the ADSP-2100 processor family, the FFT routine can produce sequentially-ordered (unscrambled) output. This method does not affect program size or execution time.

6.6.5.1 Unscrambling Routine

The unscrambling routine in Listing 6.32 uses the bit-reverse mode of the ADSP-2100 with shift operations to digit-reverse an input array of size N. The routine uses a 7-instruction core loop and unscrambles a complex pair in seven cycles. This post-unscrambling approach adds 19% to 34% more computation time to the FFT, depending on the size of the input sequence.

```
.MODULE/BOOT=0  drev_1k;

{This routine unscrambles radix-4 dif fft results out of place}
{A 7-instruction core loop unscrambles a complex pair in 7 cycles}
{An additional 20 cycles for setup are required; total processing}
{time for 1024 points is 7188 cycles or 0.57504ms @ 80ns/cycle}

.CONST          N=1024;
.CONST          Nx2=2048;
.CONST          log2N=10;
.CONST          chkrbd_e=H#2AAA;    {even bit mask}
.CONST          chkrbd_o=H#1555;    {odd bit mask}
.CONST          base_adr=H#0004;    {results base, H#0800=2K bit rev}
.CONST          evenshift=14 - log2N - 2;
.CONST          oddshift=14 - log2N;
.VAR/ABS=2048   drev_out[Nx2];      {output at 2K}
.GLOBAL         drev_out;
.EXTERNAL       inplace;
.ENTRY          drev;

drev:    ENA BIT_REV;
         M1=H#2000;         {modify by one in b_rev mode}
         M3=base_adr;       {bit-rev base adr of output buffer}
         I0=base_adr;       {initialized, for first write is wrapped}
         L0=0;
         I4=^inplace;       {I4 points to scrambled fft results}
         M4=1;
         L4=0;
         AX0=chkrbd_e;      {used to isolate even index bits}
         AX1=chkrbd_o;      {used to isolate odd index bits}
         AY0=-1;            {initialize index for wrapped code}
         SE=evenshift;      {core loop even shift is not immediate}
         AF=PASS AY0;       {store index count in AF}
         CNTR=N;            {process N complex pairs}
         DO digit UNTIL CE;
            AF=AF+1,DM(I0,M1)=MX0;  {inc index, store b_rev real val}
            AR=AX0 AND AF,DM(I0,M1)=MX1;
                        {isolate even bits, store b_rev imag}
            SR=LSHIFT AR(HI),MX0=DM(I4,M4);
                        {shift for b_rev index, get next real}
            AR=AX1 AND AF,MX1=DM(I4,M4);
                        {isolate odd bits,get imag val}
            SR=SR OR LSHIFT AR BY oddshift (HI);   {OR for b_rev index}
            I0=SR1;           {store b_rev index}
digit:      MODIFY(I0,M3);    {compute b_rev adr}
```

```
        DM(I0,M1)=MX0;       {store last b_rev real val}
        DM(I0,M1)=MX1;       {store last b_rev imag val}
        DIS BIT_REV;
        RTS;
.ENDMOD;
```

Listing 6.32 Digit-Reverse (Unscrambling) Routine

6.6.5.2 Modified Butterfly

In a radix-4 in-place algorithm, interchanging the middle two branches of every butterfly computation results in a bit-reversed output (and not a digit-reversed output). Subsequently, the ENA BIT_REV instruction (enable bit-reverse mode) of the ADSP-2100 can be invoked to bit-reverse the output sequence as it is being written out. Since the output is already in bit-reversed order, invoking the bit-reverse mode in the last stage actually puts the output in sequential order. Interchanging the middle two branches of the butterfly is depicted in Figure 6.15.

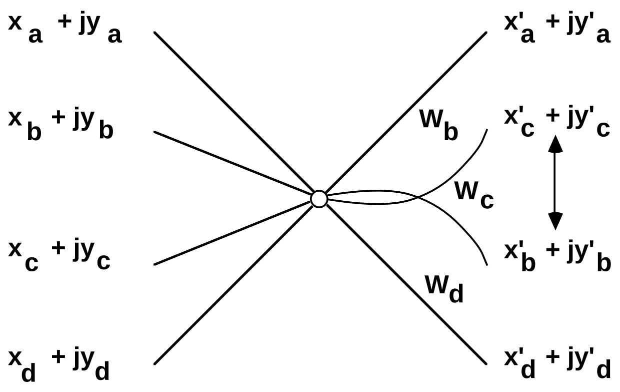

Figure 6.15 Butterfly with Interchange

In implementing this change, computations are done in the same order as before. However, in writing the results to memory we swap the middle two branches:

x_b' to position x_c', x_c' to position x_b'
y_b' to position y_c', y_c' to position y_b'

Listing 6.33 shows the butterfly code with the two branches interchanged.

```
{———————————— Stage 1 ————————————}


stage1: I0=^inplace;                             {in ->Xa,Xc}
            I1=^inplace+Nov2;                    {in+N/2 ->Xb,Xd}
            I2=^inplace+1;                       {in+1 ->Ya,Yc}
            I3=^inplace+Nov2+1;                  {in+N/2+1 ->Yb,Yd}
            I5=^cos_table;
            I6=^cos_table;
            I7=^cos_table;
            M0=N;                                {N, skip forward to dual node}
            M1=-N;                         {-N, skip back to primary node}
            M2=-N+2;                       {-N+2, skip to next butterfly}
            M5=Nov4+1;                     {N/4 + groups/stage*1, Cb Sb offset}

            M3=-2;             {Because we have modified the middle branches}
                               {of bfly, pointers for I0 have to be treated a}
                               {little more carefully. This requires a more}
                               {complex pointer manipulation, using M3.}

            M6=Nov4+2;                     {N/4 + groups/stage*2, Cc Sc offset}
            M7=Nov4+3;                     {N/4 + groups/stage*3, Cd Sc offset}
            AX0=DM(I0,M0);                       {get first Xa}
            AY0=DM(I0,M1);                       {get first Xc}
            AR=AX0-AY0, AX1=DM(I2,M0);           {Xa-Xc,get first Ya}
            SR=LSHIFT AR(LO), AY1=DM(I2,M1);     {SR1=Xa-Xc,get first Yc}
            CNTR=Nov4;                           {Bfly/group, stage one}


{Middle 2 branches of butterfly are reversed.}
{This alteration, done in every stage, results in bit-reversed}
{outputs instead of digit-reversed outputs.}


            DO stg1bfy UNTIL CE;
               AR=AX0+AY0, AX0=DM(I1,M0);        {AR=xa+xc, AX0=xb}
               MR0=AR, AR=AX1+AY1;               {MR0=xa+xc, AR=ya+yc}
               MR1=AR, AR=AX1-AY1;               {MR1=ya+yc, AR=ya-yc}
               SR=SR OR LSHIFT AR (HI), AY0=DM(I1,M1); {SR1=ya-yc, AY0=xd}
               AF=AX0+AY0, AX1=DM(I3,M0);        {AF=xb+xd, AX1=yb}
               AR=MR0+AF, AY1=DM(I3,M1);         {AR=xa+xb+xc+xd, AY1=yd}
               DM(I0,M0)=AR, AR=MR0-AF;
                               {output x’a=(xa+xb+xc+xd), AR=xa+xc-xb-xd}
               AF=AX1+AY1, MX0=AR;               {AF=yb+yd, MX0=xa+xc-xb-xd}
               AR=MR1+AF, MY0=DM(I6,M4);         {AR=ya+yc+yb+yd, MY0=(Cc)}
               DM(I2,M0)=AR, AR=MR1-AF;          {output y’a, AR=ya+yc-yb-yd}
```

```
          MR=MX0*MY0(SS), MY1=DM(I6,M6); {MR=(xa+xc-xb-xd)(Cc), MY1=(Sc)}
          MR=MR+AR*MY1(RND),SI=DM(I0,M2);
                               {MR=(xa-xb+xc-xd)(Cc)+(ya-yb+yc-yd)(Sc)}
                      {SI here is a dummy to perform a modify(I0,M2)}
          DM(I1,M0)=MR1, MR=AR*MY0(SS);    {output x'c to position x'b}
                                           {MR=(ya+yc-yb-yd)(Cc)}
          MR=MR-MX0*MY1(RND), MY0=DM(I5,M4);
                               {MR=(ya+yc-yb-yd)(Cc)-(xa+xc-xb-xd)(Sc)}
                                           {MY0=(Cb)}
          DM(I3,M0)=MR1, AR=AX0-AY0;
                               {output y'c=to position y'b, AR=xb-xd}
          AY0=AR, AF=AX1-AY1;              {AY0=xb-xd, AF=yb-yd}
          AR=SR0-AF, MY1=DM(I5,M5);        {AR=xa-xc-(yb-yd), MY1=(Sb)}
          MX0=AR, AR=SR0+AF;         {MX0=xa-xc-yb+yd, AR=xa-xc+yb-yd}
          SR0=AR, AR=SR1+AY0;        {SR0=xa-xc+yb-yd, AR=ya-yc+xb-xd}
          MX1=AR, AR=SR1-AY0;        {MX1=ya-yc+xb-xd, AR=ya-yc-(xb-xd)}
          MR=SR0*MY0(SS),AX0=DM(I0,M0);
                      {MR=(xa-xc+yb-yd)(Cb), AX0=xa of next bfly}
          MR=MR+AR*MY1(RND),AY0=DM(I0,M3);
                      {MR=(xa-xc+yb-yd)(Cb)+(ya-yc-xb+xd)(Sb)}
                                           {AY0=xc of next bfly}
          DM(I0,M2)=MR1, MR=AR*MY0(SS);
                {output x'b=to position x'c, MR=ya-yc-xb+xd)(Cb)}
          MR=MR-SR0*MY1(RND), MY0=DM(I7,M4);
                {MR=(ya-yc-xb+xd)(Cb)-(xa-xc+yb-yd)(Sb), MY0=(Cd)}
          DM(I2,M2)=MR1, MR=MX0*MY0(SS);
                {output y'b to position y'c, MR=(xa-yb-xc+yd)(Cd)}
          MY1=DM(I7,M7), AR=AX0-AY0;       {MY1=(Sd), AR=xa-xc}
          MR=MR+MX1*MY1(RND), AX1=DM(I2,M0);
                      {MR=(xa-yb-xc+yd)(Cd)+(ya+xb-yc-xd)(Sd)}
                                           {AX1=ya of next bfly}
          DM(I1,M2)=MR1, MR=MX1*MY0(SS);
                      {output x'd=(xa-yb-xc+yd)(Cd)+(ya+xb-yc-xd)(Sd)}
                                           {MR=(ya+yb-yc-yd)(Cd)}
          MR=MR-MX0*MY1(RND), AY1=DM(I2,M1);
                  {MR=(ya+yb-yc-yd)(Cd)-(xa-xc-yb+yd)Sd, yc of next bfly}
stg1bfy:  DM(I3,M2)=MR1, SR=LSHIFT AR(LO);
                      {output y'd=(ya+xb-yc-xd)Cd-(xa-xc-yb+yd)Sd}
                                           {SR0=ya-yc of next bfly}
```

Listing 6.33 Butterfly with Middle Two Branches Interchanged

Because index registers are modified after every memory access, it is important to make sure that interchanging these middle branches does not affect the remainder of the butterfly. This requires special attention to make sure that memory fetches directly following the interchanged pair are not affected.

In this new addressing sequence we define an additional modify value, M3, and use it to mask out the effect of this interchange. Figure 6.16 illustrates how, for example, index register I0 is modified through one iteration of the code in Listing 6.33. The arrow numbers correspond to the order in which I0 is sequenced.

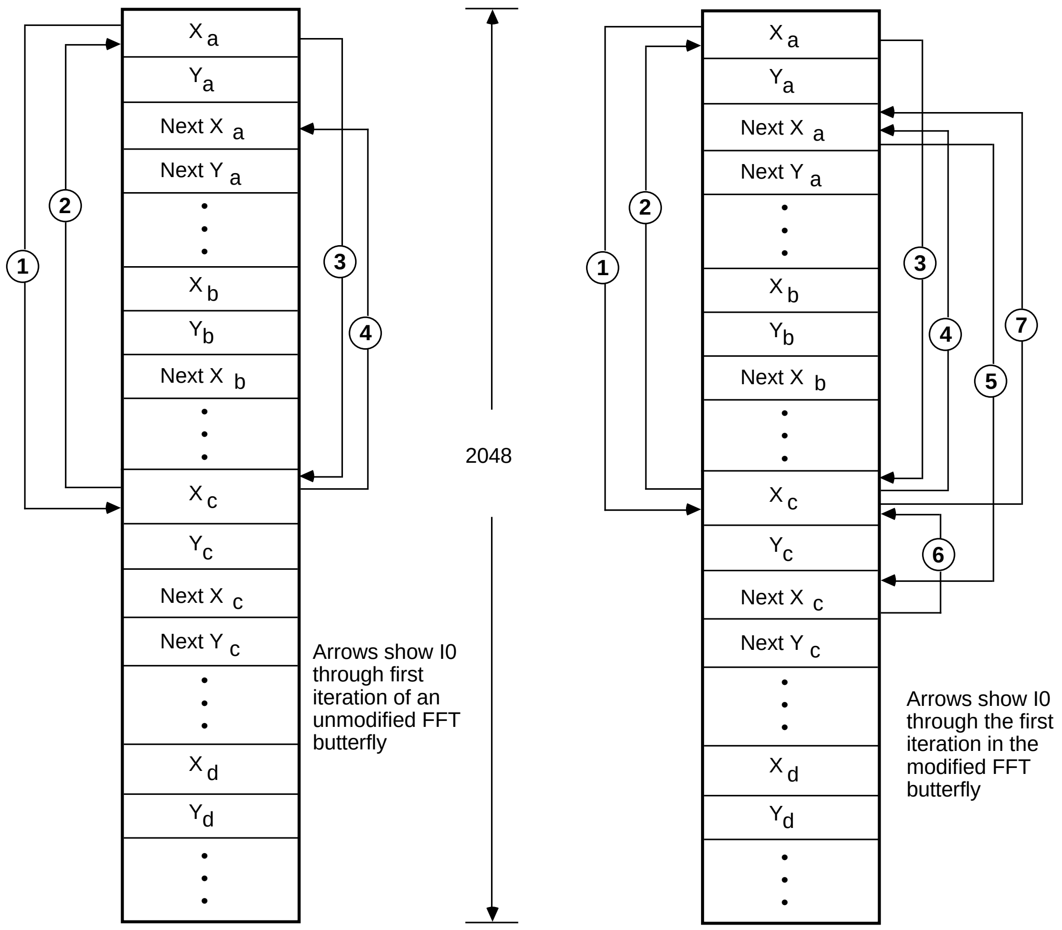
X_a
Y_a
Next X_a
Next Y_a
X_b
Y_b
Next X_b
X_c
Y_c
Next X_c
Next Y_c
X_d
Y_d
1
2
3
4
5
6
7
2048
Arrows show I0 through first iteration of an unmodified FFT butterfly
Arrows show I0 through the first iteration in the modified FFT butterfly

Figure 6.16 Modified Pointer Sequence

Since the algorithm calls two separate butterfly subroutines—one for the last stage (no multiplications) and one for the remainder of the stages—the modifications described here apply to both subroutines.

As a final step in the process, the output is sorted into sequential order using the bit-reverse mode of the ADSP-2100. This bit-reverse operation is concurrent with the execution of the last stage. The ENA BIT_REV instruction is executed at the start of the last stage. In this mode, the addresses of all memory accesses using I0, I1, I2 or I3 are bit-reversed and then placed on the address bus. Since this is done in the same clock cycle as the memory access itself, it generates no overhead.

The bit-reverse circuitry in the ADSP-2100 address generator reverses all 14 bits of the address value. Output sequences that are smaller than 2^{13} points require only $\log_2 2N$ (where N is the number of points in the FFT) bits to be reversed (2N because real and imaginary data are interleaved). For example, a 2K block of memory at H#0000 loaded with 1024 complex points, interleaved real and imaginary, in sequential order, requires reversing bits 10 through 1 of the 14-bit address value. Bit 0 is not reversed because data points are interleaved, so only every other address needs bit-reversing.

b13 b12 b11 **b10 b9 b8 b7 b6 b5 b4 b3 b2 b1** b0

becomes

b13 b12 b11 **b1 b2 b3 b4 b5 b6 b7 b8 b9 b10** b0

Bit-reversing fewer than 14 bits can be done on the ADSP-2100 by initializing I registers and an M register to appropriate values. The second column of Figure 6.17 (on the next page) shows the addresses that result from reversing bits 10 through 1 of the sequentially ordered addresses in the first column. The 14-bit addresses in the third column are the values (before bit-reverse) that the ADSP-2100 must generate in order to output the bit-reversed sequence in the second column. The addresses in the third column are obtained by initializing I0, I2, I1 and I3 to H#0000, H#2000, H#0008 and H#2008, respectively, and setting the M0 register (used to modify each of the I registers) to H#0010.

Sequentially ordered		*Bits 1 to 10 reversed*		*ADSP-2100 address (before bit-reverse)*	
data	address	data	address		
x0	00 0000 0000 0000	x0	00 0000 0000 0000	00 0000 0000 0000 = H#0000	I0
y0	00 0000 0000 0001	y0	00 0000 0000 0001	10 0000 0000 0000 = H#2000	I2
x1	00 0000 0000 0010	x512	00 0100 0000 0000	00 0000 0000 1000 = H#0008	I1
y1	00 0000 0000 0011	y512	00 0100 0000 0001	10 0000 0000 1000 = H#2008	I3
x2	00 0000 0000 0100	x256	00 0010 0000 0000	00 0000 0001 0000 = H#0010	I0+M0
y2	00 0000 0000 0101	y256	00 0010 0000 0001	10 0000 0001 0000 = H#2010	I2+M0
x3	00 0000 0000 0110	x768	00 0110 0000 0000	00 0000 0001 1000 = H#0018	I1+M0
y3	00 0000 0000 0111	y768	00 0110 0000 0001	10 0000 0001 1000 = H#2018	I3+M0
•		•		•	
•		•		•	
•		•		•	

Figure 6.17 Reversing Bits 10 Through 1

6.6.6 Variations

6.6.6.1 Inverse FFT

The inverse relationship for obtaining a sequence from its DFT is called the inverse DFT (IDFT). The transformation is described by the equation:

$$x(n) \ = 1/N \quad \sum_{k=0}^{N-1} X(k)\, W_N^{-nk}$$

Although the FFT algorithms described in the chapter were presented in the context of computing the DFT efficiently, they may also be used in computing the IDFT.

The only difference between the two transformations is the normalization factor 1/N and the phase sign of the twiddle factor W_N. Consequently, an FFT algorithm for computing the DFT may be converted into an IFFT algorithm for computing the IDFT by using a reversed (upside down) twiddle factor table and by dividing the output of the algorithm by N.

6.6.6.2 Sizing the Program
Thus far, the discussion has been directed at the 1024-point FFT. It is possible with some changes to use the same code to transform 4s samples, where s is the number of stages required to do that. The modifications to resize the FFT program are as follows:

- Create input data files of new size.
- Generate twiddle factor table of new size.
- Modify symbolic constants in all modules used,
- Add/delete stages as required to satisfy the relation log4N=number of stages. Because the first and last stages are optimized for speed, additions or deletions are done to the center of the program.

In the program, all the stages with the exception of the optimized first and last have the same structure (see Listing 6.31). First, several instructions initialize the various registers, pointers and counters. Next, a subroutine that executes the group and butterfly loops is called and all the computations for that stage are carried out.

In reducing the size of the FFT, an entire stage block is deleted and the initial values of the remaining stages are recomputed using the new size. (The specific initial values for each stage as a function of size are documented in Listing 6.31.)

If the new size of the FFT is more than 1024 points, additional stages are required. This means another stage is written with setup instructions and a subroutine call identical to the other stages. This new stage is inserted in the middle of the FFT, and the initialization values for all the stages of the new program are recomputed following the documentation in Listing 6.31.

6.6.7 Programs and File Description
This section presents the files and the programs used in the optimized FFT example of this chapter. The flowchart below illustrates how the various files and modules are interrelated (see Figure 6.18 on the next page). Data files are shown in ovals and operations in rectangles.

6.6.7.1 Twiddle Factors
TWIDDLES.C is a C program that generates the data file for the cosine table (twiddle factor table). This hexadecimal data file (COS1024.DAT) is fully normalized in 1.15 format. The data on this file is loaded into memory through the .INIT directive during the linking process.

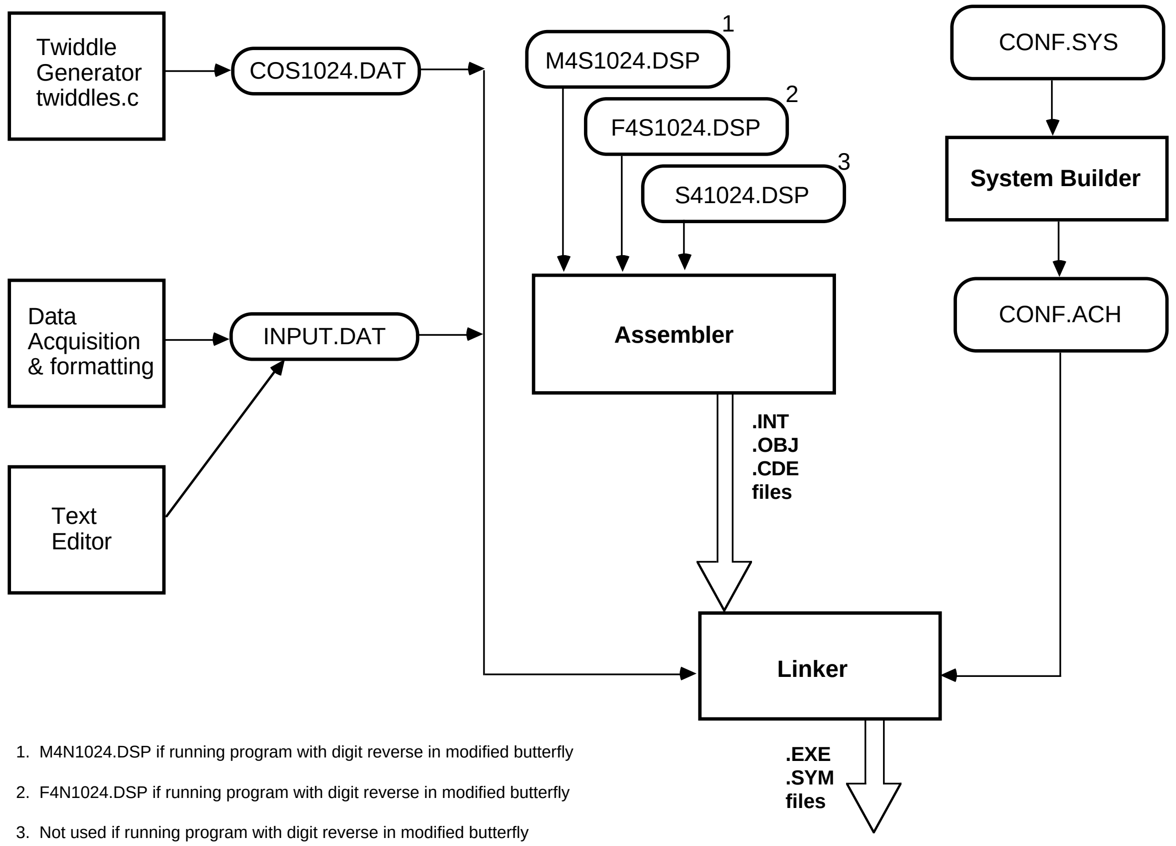
Twiddle Generator twiddles.c
COS1024.DAT
M4S1024.DSP
1
F4S1024.DSP
2
S41024.DSP
3
CONF.SYS
System Builder
CONF.ACH
Data Acquisition & formatting
INPUT.DAT
Assembler
Text Editor
.INT
.OBJ
.CDE
files
Linker
.EXE
.SYM
files
1. M4N1024.DSP if running program with digit reverse in modified butterfly
2. F4N1024.DSP if running program with digit reverse in modified butterfly
3. Not used if running program with digit reverse in modified butterfly

Figure 6.18 Program Generation Flowchart

6.6.7.2 Input Data

INPUT.DAT is a formatted (1.15 format, 11 guard bits) hexadecimal data file that contains the 1024 complex input samples (interleaved real and imaginary). The file is loaded into the *inplace* array through the .INIT directive during the linking process. In a real-time operation, the input samples would be loaded into RAM directly via a data acquisition board.

6.6.7.3 FFT Routines

The files F4S1024.DSP and F4N1024.DSP each contain a module fft that performs the FFT computations and forms the heart of the program. The module in F4S1024.DSP is called by the main routine in M4S1024.DSP and produces scrambled results which are passed to the *drev* routine in S41024.DSP to be unscrambled. The module in F4N1024.DSP is called by the main routine in M4N1024.DSP and produces normal (sequentially ordered) output at addresses H#0000-07FF.

6.6.7.4 FFT Program with Unscrambling Routine

Listing 6.34 shows the main module for the optimized radix-4 FFT, written for the ADSP-2101. This module is stored in the file M4S1024.DSP, which is the calling shell for the *fft* routine (in file F4S1024.DSP) and the *drev* unscrambling routine (in file S41024.DSP). The restart value for the PC is 0 and therefore the code starts immediately. The first three instructions

```
   SI=0;
   DM(H#3FFF)=SI;
   DM(H#3FFE)=SI;
```

reset the system control register and the memory control register to allow zero wait state memory access (defaults to seven wait states).

```
.MODULE/RAM/BOOT=0/ABS=0            main;

{
Calling shell for the 1024-point radix-4 FFT with the unscrambling routine.

            fft = FFT routine (in F4S1024.DSP)
            drev = unscrambling routine (in S41024.DSP)
}

.CONST      N=1024;
.CONST      Nx2=2048;
.VAR/DM     inplace[Nx2];     {inplace array contains original input}
                              {and also holds intermediate results}
.INIT       inplace:<input.dat>;    {load inplace array with data}
.GLOBAL     inplace;
.EXTERNAL   fft,drev;               {2 routines used to perform FFT}



            SI=0;                   {These 3 lines reset the system}
            DM(H#3FFF)=SI;          {control register and the data}
            DM(H#3FFE)=SI;          {memory control register to allow}
                                    {zero state memory access}
            CALL fft;
            CALL drev;

trapper:    JUMP trapper;

.ENDMOD;
```

Listing 6.34 Main Module (ADSP-2101) for Radix-4 DIF FFT with Unscrambling Routine

In implementing the FFT on the ADSP-2100, the main module is modified in the following way. The ADSP-2100 has a PC restart value of 4 and an interrupt vector table in the 4-word address space 0000-0003. Therefore, the sequence of instructions below is required at the start of the module. The four RTI instructions ensure a NOP in case of a false interrupt signal.

```
   RTI;   In the ADSP-2100 the first 4 PM locations
   RTI;   are reserved for the interrupt vector
   RTI;   table.
   RTI;
< code starts here for ADSP-2100 >
```

The ADSP-2100 has no programmable wait states, so the first three instructions in Listing 6.34 should be removed. In addition, since the ADSP-2100 has no on-chip memory, the boot page select directive / BOOT=0 is not valid and should be removed.

6.6.7.5 FFT Program with Built-In Digit Reversal

Listing 6.35 contains the main routine for the optimized radix-4 FFT with built-in digit-reversal. Listing 6.36 contains the fft routine called by the main routine. Note that the unscrambling routine is not required here.

```
.MODULE/RAM/BOOT=0/ABS=0   main;

{Calling shell for 1024-point radix-4 FFT with built-in digit-reverse.

   fft = FFT routine (in F4N1024.DSP)}

.CONST              N=1024;           {number of samples in FFT}
.CONST              Nx2=2048;
.VAR/DM             inplace[Nx2]; {inplace array contains original input}
                                {and also holds intermediate results}
.VAR/DM/ABS=0       output[Nx2]; {output array holds results in order}
.INIT               inplace:<input.dat>;{load inplace array with data}
.GLOBAL             inplace,output;
.EXTERNAL           fft;              {FFT routine}

                    SI=0;             {these 3 lines reset the system}
                    DM(H#3FFF)=SI;    {control register and the data}
                    DM(H#3FFE)=SI;    {memory control register to allow}
                                      {zero wait state memory access}
                    CALL fft;
trapper:            JUMP trapper;

.ENDMOD;
```

Listing 6.35 Main Module for Radix-4 DIF FFT with Built-In Digit-Reversal

```
.MODULE/BOOT=0   fft_sub;

{
Optimized complex 1024-point radix-4 DIF FFT

This routine uses a modified radix-4 algorithm to unscramble results as
they are computed. The results are thus in sequential order.

Complex data is stored as x0 (real, imag), y0 (real, imag), x1 (real,
imag), y1 (real, imag),...

Butterfly terms
   xa = 1st real input leg        x’a = 1st real output leg
   xb = 2nd real input leg        x’b = 2nd real output leg
   xc = 3rd real input leg        x’c = 3rd real output leg
   xd = 4th real input leg        x’d = 4th real output leg
   ya = 1st imag input leg        y’a = 1st imag output leg
   yb = 2nd imag input leg        y’b = 2nd imag output leg
   yc = 3rd imag input leg        y’c = 3rd imag output leg
   yd = 4th imag input leg        y’d = 4th imag output leg

Pointers
   I0 —> xa,xc
   I1 —> xb,xd
   I2 —> ya,yc
   I3 —> yb,yd
   w0 (= Ca = Sa = 0)
   I5 —> w1 (1st Cb, - pi/4 for Sb)
   I6 —> w2 (2nd Cc, - pi/4 for Sc)
   I7 —> w3 (3rd Cd, - pi/4 for Sd)

Input
   inplace[2*N] normal order, interleaved real, imag.

Output
   inplace[2*N] normal order, interleaved real, imag.

Computation Time (cycles)
   setup   = 9
   stage 1 = 7700 = 20+256(30)
   stage 2 = 7758 = 18+4(15+64(30))
   stage 3 = 7938 = 18+16(15+16(30))
   stage 4 = 8658 = 18+64(15+4(30))
   stage 5 = 5140 = 20+256(20)

   Total 37203 cycles * 80ns/cycle = 2.97624ms}
```

(listing continues on next page)

```
.CONST          N = 1024;
.CONST          NX2 = 2048;
.CONST          Nov2 = 512;
.CONST          Nov4 = 256;
.CONST          N3ov8 = 384;
.VAR/DM/CIRC    cos_table[1024];

.VAR/DM         m3_space;   {memory space used to store M3 values}
                            {when M3 is loaded with its alternate value}

.VAR/DM         bfy_count;
.INIT           cos_table:<cos1024.dat>;   {N cosine values}
.EXTERNAL       inplace;

.ENTRY          fft;

fft:            M4=-Nov4;      {-N/4 = -90 degrees for sine}
                L0=0;
                L1=0;
                L2=0;
                L3=0;
                L5=%cos_table;
                L6=%cos_table;
                L7=%cos_table;
                SE=0;

{——————————— Stage 1 ———————————}

stage1:     I0=^inplace;             {in ->Xa,Xc}
            I1=^inplace+Nov2;        {in+N/2 ->Xb,Xd}
            I2=^inplace+1;           {in+1 ->Ya,Yc}
            I3=^inplace+Nov2+1;      {in+N/2+1 ->Yb,Yd}
            I5=^cos_table;
            I6=^cos_table;
            I7=^cos_table;
            M0=N;                    {N, skip forward to dual node}
            M1=-N;                   {-N,   skip back to primary node}
            M2=-N+2;                 {-N+2, skip to next butterfly}

            M3=-2;      {Because we have modified the middle branches}
                        {of bfly, pointers for I0 require more}
                        {complex manipulation, using M3}

            M5=Nov4+1;               {N/4 + groups/stage*1, Cb Sb offset}
            M6=Nov4+2;               {N/4 + groups/stage*2, Cc Sc offset}
            M7=Nov4+3;               {N/4 + groups/stage*3, Cd Sc offset}
            AX0=DM(I0,M0);          {get first Xa}
            AY0=DM(I0,M1);          {get first Xc}
            AR=AX0-AY0, AX1=DM(I2,M0);    {Xa-Xc,get first Ya}
            SR=LSHIFT AR(LO), AY1=DM(I2,M1); {SR1=Xa-Xc,get first Yc}
            CNTR=Nov4;              {Bfly/group, stage one}

{Middle 2 branches of butterfly are reversed.}
{This alteration, done in every stage, results in bit-reversed}
{outputs instead of digit-reversed outputs.}


            DO stg1bfy UNTIL CE;
               AR=AX0+AY0, AX0=DM(I1,M0); {AR=xa+xc, AX0=xb}
               MR0=AR, AR=AX1+AY1;        {MR0=xa+xc, AR=ya+yc}
               MR1=AR, AR=AX1-AY1;        {MR1=ya+yc, AR=ya-yc}
               SR=SR OR LSHIFT AR (HI), AY0=DM(I1,M1);
                                          {SR1=ya-yc, AY0=xd}
               AF=AX0+AY0, AX1=DM(I3,M0); {AF=xb+xd, AX1=yb}
               AR=MR0+AF, AY1=DM(I3,M1);  {AR=xa+xb+xc+xd, AY1=yd}
               DM(I0,M0)=AR, AR=MR0-AF;
                              {output x’a=(xa+xb+xc+xd), AR=xa+xc-xb-xd}
               AF=AX1+AY1, MX0=AR;        {AF=yb+yd, MX0=xa+xc-xb-xd}
               AR=MR1+AF, MY0=DM(I6,M4);  {AR=ya+yc+yb+yd, MY0=(Cc)}
               DM(I2,M0)=AR, AR=MR1-AF;   {output y’a, AR=ya+yc-yb-yd}
               MR=MX0*MY0(SS), MY1=DM(I6,M6);
                                    {MR=(xa+xc-xb-xd)(Cc), MY1=(Sc)}
               MR=MR+AR*MY1(RND),SI=DM(I0,M2);
                              {MR=(xa-xb+xc-xd)(Cc)+(ya-yb+yc-yd)(Sc)}
                          {SI here is a dummy to perform a modify(I0,M2)}
               DM(I1,M0)=MR1, MR=AR*MY0(SS);
                                          {output x’c to position x’b}
                                          {MR=(ya+yc-yb-yd)(Cc)}
               MR=MR-MX0*MY1(RND), MY0=DM(I5,M4);
                       {MR=(ya+yc-yb-yd)(Cc)-(xa+xc-xb-xd)(Sc), MY0=(Cb)}
               DM(I3,M0)=MR1, AR=AX0-AY0;
                              {output y’c=to position y’b, AR=xb-xd}
               AY0=AR, AF=AX1-AY1;        {AY0=xb-xd, AF=yb-yd}
               AR=SR0-AF, MY1=DM(I5,M5);  {AR=xa-xc-(yb-yd), MY1=(Sb)}
               MX0=AR, AR=SR0+AF; {MX0=xa-xc-yb+yd, AR=xa-xc+yb-yd}
               SR0=AR, AR=SR1+AY0; {SR0=xa-xc+yb-yd, AR=ya-yc+xb-xd}
               MX1=AR, AR=SR1-AY0; {MX1=ya-yc+xb-xd, AR=ya-yc-(xb-xd)}
               MR=SR0*MY0(SS),AX0=DM(I0,M0);
                          {MR=(xa-xc+yb-yd)(Cb), AX0=xa of next bfly}
               MR=MR+AR*MY1(RND),AY0=DM(I0,M3);
                              {MR=(xa-xc+yb-yd)(Cb)+ (ya-yc-xb+xd)(Sb)}
                                          {AY0=xc of next bfly}
```

(listing continues on next page)

```
               DM(I0,M2)=MR1, MR=AR*MY0(SS);
                     {output x’b to position x’c, MR=ya-yc-xb+xd)(Cb)}
               MR=MR-SR0*MY1(RND), MY0=DM(I7,M4);
                     {MR=(ya-yc-xb+xd)(Cb)-(xa-xc+yb-yd)(Sb), MY0=(Cd)}
               DM(I2,M2)=MR1, MR=MX0*MY0(SS);
                     {output y’b to position y’c, MR=(xa-yb-xc+yd)(Cd)}
               MY1=DM(I7,M7), AR=AX0-AY0; {MY1=(Sd), AR=xa-xc}
               MR=MR+MX1*MY1(RND), AX1=DM(I2,M0);
                     {MR=(xa-yb-xc+yd)(Cd)+(ya+xb-yc-xd)(Sd)}
                                            {AX1=ya of next bfly}
               DM(I1,M2)=MR1, MR=MX1*MY0(SS);
                     {output x’d=(xa-yb-xc+yd)(Cd)+(ya+xb-yc-xd)(Sd)}
                                            {MR=(ya+yb-yc-yd)(Cd)}
               MR=MR-MX0*MY1(RND), AY1=DM(I2,M1);
                              {MR=(ya+yb-yc-yd)(Cd)-(xa-xc-yb+yd)Sd}
                                            {AY1=yc of next bfly}
stg1bfy:    DM(I3,M2)=MR1, SR=LSHIFT AR(LO);
                        {output y’d=(ya+xb-yc-xd)Cd-(xa-xc-yb+yd)Sd}
                                            {SR0=ya-yc of next bfly}

{———————————— Stage 2 ————————————}

stage2:     I0=^inplace;              {in -> Xa,Xc}
            I1=^inplace+128;          {in+N/8 -> Xb,Xd}
            I2=^inplace+1;            {in+1 -> Ya,Yc}
            I3=^inplace+129;          {in+N/8+1 -> Yb,Yd}
            M0=Nov4;                  {N/4, skip forward to dual node}
            M1=-Nov4;                 {-N/4, skip back to primary node}
            M2=-Nov4+2;               {-N/4+2, skip to next butterfly}

            M3=384;                   {N*3/8, skip to next group}
            DM(m3_space)=M3;          {m3_space is temporary storage}
                                      {space needed because M3 is used}
                                      {in 2 contexts and will alternate}
                                      {in value}

            M5=Nov4+4;                {N/4+groups/stage*1, Cb Sb offset}
            M6=Nov4+8;                {N/4+groups/stage*2, Cc Sc offset}
            M7=Nov4+12;               {N/4+groups/stage*3, Cd Sd offset}
            SI=64;              {Bfy/group, save counter for inner loop}
            DM(bfy_count)=SI; {SI is used as a temporary storage dummy}

            CNTR=4;                   {groups/stage}
            CALL mid_stg;             {do stage 2}
```

```
{———————————— Stage 3 ————————————}

stage3:     I0=^inplace;            {in -> Xa,Xc}
            I1=^inplace+32;         {in+N/32 -> Xb,Xd}
            I2=^inplace+1;          {in+1 -> Ya,Yc}
            I3=^inplace+33;         {in+N/32+1 -> Yb,Yd}
            M0=64;                  {N/16, skip forward to dual node}
            M1=-64;                 {-N/16, skip back to primary node}
            M2=-62;                 {-N/16+2, skip to next butterfly}
            M3=96;                  {N*3/32, skip to next group}

            DM(m3_space)=M3;        {M3_space is temporary storage}
                                    {space needed because M3 is used}
                                    {in 2 contexts and will alternate}
                                    {in value}

            M5=Nov4+16;             {N/4+groups/stage*1, Cb Sb offset}
            M6=Nov4+32;             {N/4+groups/stage*2, Cc Sc offset}
            M7=Nov4+48;             {N/4+groups/stage*3, Cd Sd offset}
            SI=16;            {Bfly/group, save counter for inner loop}
            DM(bfy_count)=SI;       {SI is a dummy temporary register}

            CNTR=16;                {groups/stage}
            CALL mid_stg;           {do stage 3}

{———————————— Stage 4 ————————————}

stage4:     I0=^inplace;            {in -> Xa,Xc}
            I1=^inplace+8;          {in+N/128 -> Xb,Xd}
            I2=^inplace+1;          {in+1 -> Ya,Yc}
            I3=^inplace+9;          {in+N/128+1 -> Yb,Yd}
            M0=16;                  {N/64, skip forward to dual node}
            M1=-16;                 {-N/64, skip back to primary node}
            M2=-14;                 {-N/64+2, skip to next butterfly}

            M3=24;                  {N*3/128, skip to next group}
            DM(m3_space)=M3;        {M3_space is temporary storage}
                                    {space needed because M3 is used}
                                    {in 2 contexts and will alternate}
                                    {in value}

            M5=Nov4+64;             {N/4+groups/stage*1, Cb Sb offset}
            M6=Nov4+128;            {N/4+groups/stage*2, Cc Sc offset}
            M7=Nov4+192;            {N/4+groups/stage*3, Cd Sd offset}
            SI=4;                {Bfly/group, save counter inner loop}
            DM(bfy_count)=SI;       {SI is a dummy used for storage}
```

(listing continues on next page)

```
            CNTR=64;                {groups/stage}
            CALL mid_stg;           {do stage 4}

{———————— Last Stage, No Multiplies ————————}

laststage:  I4=^inplace;            {in ->Xa,Xc}
            I5=^inplace+2;          {in+N/512 ->Xb,Xd}
            I6=^inplace+1;          {in+1 ->Ya,Yc}
            I7=^inplace+3;          {in+N/512+1 ->Yb,Yd}
            M4=4;                   {N/256, skip forward to dual node}

            M0=H#0010;     {This modify value is used to perform bit-}
                           {reverse as the final results are written}
                           {out. The derivation of this value}
                           {is explained in the text.}

            I0=H#0000;         {These base address values are derived}
            I2=H#2000;         {for output at address 0000}
            I1=H#0008;
            I3=H#2008;

            L4=0;             {This last stage has no twiddle factor}
            L5=0;             {multiplication}
            L6=0;             {Because the output addresses are bit-}
            L7=0;          {reversed, the I’s M’s & L’s are reassigned}
                              {and reinitialized}

            AX0=DM(I4,M4);          {first Xa}
            AY0=DM(I4,M4);          {first Xc}
            CNTR=Nov4;              {groups/stage}
            ENA BIT_REV;            {all data accesses using I0..I3 are}
                                    {bit-reversed}


{Middle 2 branches of butterfly are reversed.}
{This alteration, done in every stage, results in bit-reversed}
{outputs instead of digit-reversed outputs.}

            DO laststgbfy UNTIL CE;
               AR=AX0-AY0, AX1=DM(I6,M4); {AR=xa-xc, AX1=ya}
               SR=LSHIFT AR(LO), AY1=DM(I6,M4); {SR0=xa-xc, AY1=yc}
               AR=AX0+AY0, AX0=DM(I5,M4); {AR=xa+xc, AX0=xb}
               MR0=AR, AR=AX1+AY1;        {MR0=xa+xc, AR=ya+yc}
               MR1=AR, AR=AX1-AY1;        {MR1=ya+yc, AR=ya-yc}
               SR=SR OR LSHIFT AR (HI), AY0=DM(I5,M4);
                                    {SR1=ya-yc, AY0 xd}
               AF=AX0+AY0, AX1=DM(I7,M4); {AF=xb+xd, AX1=yb}
               AR=MR0+AF, AY1=DM(I7,M4);  {AR=xa+xc+xb+xd, AY1=yd}
               DM(I0,M0)=AR, AR=MR0-AF;
                              {output x'a=xa+xc+xb+xd, AR=xa+xc-(xb+xd)}
               DM(I1,M0)=AR, AF=AX1+AY1;
                              {output x'c to position x'b, AF=yb+yd}
               AR=MR1+AF;                 {AR=ya+yb+yc+yd}
               DM(I2,M0)=AR, AR=MR1-AF;
                              {output y'a=ya+yc+yb+yd, AR=ya+yc-(yb+yd)}
               DM(I3,M0)=AR, AR=AX0-AY0;
                              {output y'c to position y'b, AR=xb-xd}
               AX0=DM(I4,M4);             {AX0=xa of next group}
               AF=AX1-AY1, AY1=AR;        {AF=yb-yd, AY1=xb-xd}
               AR=SR0+AF, AY0=DM(I4,M4);  {AR=xa-xc+yb-yd}
                                          {AY0=xc of next group}
               DM(I0,M0)=AR, AR=SR0-AF;
                        {output x'b to position x'c, AR=xa-xc-(yb-yd)}
               DM(I1,M0)=AR, AR=SR1-AY1;
                        {output x'd=xa-xc-(yb-yd), AR=ya-yc+(xb-xd)}
               DM(I2,M0)=AR, AR=SR1+AY1;
                        {output y'b to position y'c, AR=ya-yc-(xb-xd)}
laststgbfy: DM(I3,M0)=AR;           {output y'd=ya-yc-(xb-xd)}

            DIS BIT_REV;            {shut-off bit reverse mode}

            RTS;                    {end and exit from FFT subroutine}

{———————— Subroutine for middle stages ————————}


mid_stg:    DO midgrp UNTIL CE;
               I5=^cos_table;
               I6=^cos_table;
               I7=^cos_table;
               AX0=DM(I0,M0);             {get first Xa}
               AY0=DM(I0,M1);             {get first Xc}
               AR=AX0-AY0, AX1=DM(I2,M0); {Xa-Xc,get first Ya}
               SR=LSHIFT AR (LO), AY1=DM(I2,M1);{SR1=Xa-Xc,get first Yc}
               CNTR=DM(bfy_count);        {butterflies/group}

               M3=-2;                     {M3 is loaded with the value}
                                        {required for pointer manipulation}

{Middle 2 branches of butterfly are reversed.}
{This alteration, done in every stage, results in bit-reversed}
{outputs instead of digit-reversed outputs.}
```

(listing continues on next page)

```
DO midbfy UNTIL CE;
   AR=AX0+AY0, AX0=DM(I1,M0); {AR=xa+xc, AX0=xb}
MR0=AR, AR=AX1+AY1;        {MR0=xa+xc, AR=ya+yc}
MR1=AR, AR=AX1-AY1;        {MR1=ya+yc, AR=ya-yc}
SR=SR OR LSHIFT AR (HI), AY0=DM(I1,M1);
                           {SR1=ya-yc, AY0=xd}
AF=AX0+AY0, AX1=DM(I3,M0); {AF=xb+xd, AX1=yb}
AR=MR0+AF, AY1=DM(I3,M1);  {AR=xa+xb+xc+xd, AY1=yd}
DM(I0,M0)=AR, AR=MR0-AF;
             {output x’a=(xa+xb+xc+xd), AR=xa+xc-xb-xd}
AF=AX1+AY1, MX0=AR;        {AF=yb+yd, MX0=xa+xc-xb-xd}
AR=MR1+AF, MY0=DM(I6,M4);  {AR=ya+yc+yb+yd, MY0=(Cc)}
DM(I2,M0)=AR, AR=MR1-AF;   {output y’a, AR=ya+yc-yb-yd}
MR=MX0*MY0(SS), MY1=DM(I6,M6);
                     {MR=(xa+xc-xb-xd)(Cc), MY1=(Sc)}
MR=MR+AR*MY1(RND), SI=DM(I0,M2);
               {MR=(xa-xb+xc-xd)(Cc)+(ya-yb+yc-yd)(Sc)}
               {SI is a dummy to cause a modify(I0,M2)}
DM(I1,M0)=MR1, MR=AR*MY0(SS);
      {output x’c to position x’b, MR=(ya+yc-yb-yd)(Cc)}
MR=MR-MX0*MY1(RND), MY0=DM(I5,M4);
         {MR=(ya+yc-yb-yd)(Cc)-(xa+xc-xb-xd)(Sc)}
                           {MY0=(Cb)}
DM(I3,M0)=MR1, AR=AX0-AY0;
               {output y’c to position y’b, AR=xb-xd}
AY0=AR, AF=AX1-AY1;        {AY0=xb-xd, AF=yb-yd}
AR=SR0-AF, MY1=DM(I5,M5);  {AR=xa-xc-(yb-yd), MY1=(Sb)}
MX0=AR, AR=SR0+AF;   {MX0=xa-xc-yb+yd, AR=xa-xc+yb-yd}
SR0=AR, AR=SR1+AY0;  {SR0=xa-xc+yb-yd, AR=ya-yc+xb-xd}
MX1=AR, AR=SR1-AY0;  {MX1=ya-yc+xb-xd, AR=ya-yc-(xb-xd)}
MR=SR0*MY0(SS), AX0=DM(I0,M0);
            {MR=(xa-xc+yb-yd)(Cb), AX0=xa of next bfly}
MR=MR+AR*MY1(RND), AY0=DM(I0,M3);
            {MR=(xa-xc+yb-yd)(Cb)+(ya-yc-xb+xd)(Sb)}
                           {AY0=xc of next bfly}
DM(I0,M2)=MR1, MR=AR*MY0(SS);
                           {output x’b to position x’c}
                           {MR=ya-yc-xb+xd)(Cb)}
MR=MR-SR0*MY1(RND), MY0=DM(I7,M4);
                {MR=(ya-yc-xb+xd)(Cb)-(xa-xc+yb-yd)(Sb)}
                           {MY0=(Cd)}
DM(I2,M2)=MR1, MR=MX0*MY0(SS);
                           {output y’b to position y’c}
                           {MR=(xa-yb-xc+yd)(Cd)}
MY1=DM(I7,M7), AR=AX0-AY0; {MY1=(Sd), AR=xa-xc}
MR=MR+MX1*MY1(RND), AX1=DM(I2,M0);
               {MR=(xa-yb-xc+yd)(Cd)+(ya+xb-yc-xd)(Sd)}
                           {AX1=ya of next bfly}
               DM(I1,M2)=MR1, MR=MX1*MY0(SS);
                        {output x’d=(xa-yb-xc+yd)(Cd)+(ya+xb-yc-xd)(Sd)}
                                          {MR=(ya+yb-yc-yd)(Cd)}
               MR=MR-MX0*MY1(RND), AY1=DM(I2,M1);
                              {MR=(ya+yb-yc-yd)(Cd)-(xa-xc-yb+yd)Sd}
                                          {yc of next bfly}
midbfy:        DM(I3,M2)=MR1, SR=LSHIFT AR(LO);
                           {output y’d=(ya+xb-yc-xd)Cd-(xa-xc-yb+yd)Sd}
                                          {SR0=ya-yc of next bfly}

            M3=DM(m3_space);     {modifier M3 is loaded with skip to}
                                 {next group_count and is used in the}
                                 {next four instructions}

            MODIFY (I0,M3);
            MODIFY (I1,M3);               {point to next group}
            MODIFY (I2,M3);               {of butterflies}
midgrp:     MODIFY (I3,M3);

            RTS;                 {return to middle stage calling code}

.ENDMOD;
```

Listing 6.36 Radix-4 DIF FFT Module with Built-In Digit-Reversal