เอกสารอ้างอิง

1. บัณฑิต โรจน์อารยานนท์, หลักการไฟฟ้าสื่อสาร ,สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย, 2532
2. Pramode K. Vernua , ISDN SYSTEMS ,Prentice-Hall ,1990
3. R.M.Gray , VECTOR QUANTIZATION ,IEEE ASSP Magazine, April 1984
4. Clarke, R.J., Transform Coding of Image, Academic Press, Florida, 1985.
5. W.Chen and W.K.Pratt, SCENE ADAPTIVE CODER ,IEEE Transactions on Communications ,March 1984
6. Texas Instruments, TMS32010 User's Guide, 1983.
7. Eggebrecht, L.C., Interfacing to the IBM Personal Computer, Howard W. Sams, Inc. Co., Indianapolis, 1983.

ภาคผนวก

ภาคผนวก ก

โปรแกรมจำลองแบบการทำงานของระบบ

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <io.h>
#include <conio.h>
#include <dos.h>
#include <fcntl.h>
#include <math.h>
#include <mem.h>
#include "\image\process\dsplab.h"
#include "\image\process\dsplab.def"

#define  DATA_SIZE          256
#define  TH_DEFAULT           4
#define  TH_HIGH             16
#define  TH_LOW               4
#define  MINLEVEL           -32
#define  MAXLEVEL            31
#define  DC_INIT           1024

struct TABLE {
       BYTE bit;
       BYTE code;
};
struct BLOCK_8BIT {
       char      pixel[BLOCK_SIZE];
};
void loadpic(char *s);
void conv_to_smallpic(struct BLOCK **pb);
void dct2_d(struct BLOCK **source,struct BLOCK **dest);
void idct2_d(struct BLOCK **source,struct BLOCK **dest);
void expand(BYTE *temp);
void clearpredictor(struct BLOCK **t_pred,struct BLOCK **r_pred,BYTE *tblock,BYTE *rblock);
unsigned bitcount(struct BLOCK **dest,BYTE *th);
void dpcm(struct BLOCK **source,struct BLOCK **pred,struct BLOCK **dest);
char quantize(int data);
void update_tpredictor(struct BLOCK **qb,struct BLOCK **pred);
void threshold_encode(struct BLOCK **source,struct BLOCK_8BIT **dest,BYTE *th);
```

```
void threshold_decode(struct BLOCK_8BIT **source,struct BLOCK **dest,BYTE *th);
void idpcm(struct BLOCK **source,struct BLOCK **pred);
void update_rpredictor(struct BLOCK **qb,struct BLOCK **pred);
void copyblock(struct BLOCK **source,struct BLOCK **dest);
void histogram(struct BLOCK **pb);
void changerate(void);
unsigned huffman_encode(struct BLOCK_8BIT **pb,BYTE *ch,BYTE *th);
void huffman_decode(BYTE *ch,struct BLOCK_8BIT **pb,BYTE *th);
void changethreshold(void);
void histogram_update(struct BLOCK_8BIT **pb,unsigned long *hist,unsigned long *zero);
void histogram_save(unsigned long *hist,unsigned long *zero);
void save_pic();
void save_channel(BYTE *ch);
unsigned sgxv_encode(struct BLOCK_8BIT **pb);

int bound = 32;
int thset = 32;
struct TABLE huffman[14] = {  {0,0},
                              {2,0x01},
                              {4,0x01},
                              {5,0x07},
                              {6,0x01},
                              {6,0x0D},
                              {7,0x19},
                              {8,0x01},
                              {8,0x31},
                              {9,0x00},
                              {9,0x60},
                              {9,0x01},
                              {9,0x61},
                              {7,0x01} };
struct TABLE run_length[31] = { {0,0},
                                {2,0x03},
                                {3,0x05},
                                {3,0x03},
                                {4,0x05},
                                {4,0x03},
                                {5,0x08},
```

```
                                {5,0x12},
                                {5,0x09},
                                {5,0x11},
                                {5,0x13},
                                {6,0x08},
                                {6,0x20},
                                {6,0x0A},
                                {6,0x09},
                                {6,0x21},
                                {6,0x03},
                                {6,0x0B},
                                {7,0x00},
                                {7,0x04},
                                {7,0x02},
                                {7,0x0E},
                                {7,0x01},
                                {7,0x05},
                                {7,0x03},
                                {7,0x0F},
                                {8,0x18},
                                {8,0x1A},
                                {8,0x19},
                                {8,0x1B},
                                {11,0x02} };
int main()
{
      int  i,j,count,nexttth,nextst,c;
      char s[5],picname[30],*pname,bank;
      unsigned bit_count = 0,xvbit_count = 0;
      unsigned long size,bits,hist[33],zero[64],xvbits;
      struct  BLOCK   **pblock,**tblock,**qblock,**t_predictor,**r_predictor,**rtblock,**tb,
**dct_block,**idct_block;
      struct BLOCK_8BIT **sblock,**rblock,**pb8;
      BYTE *spic,*temp,*channel,*dest,*diff;
      BYTE *t_th_block,*r_th_block;
      double avr,snr;
      BYTE processed = FALSE;
```

```
        initial();
        setmem(hist,sizeof(unsigned long) * 33,0);
        setmem(zero,sizeof(unsigned long) * 64,0);
        channel = farmalloc(sizeof(BYTE) * 65535);  /* 64 kbits */
        if(!channel) {
          cputs("Not enough memory channel!\n\r");
          exit(1);
        }
        t_th_block = farmalloc(sizeof(BYTE) * DATA_SIZE);
        if(!t_th_block) {
          cputs("Not enough memory t_th_block!\n\r");
          exit(1);
        }
        r_th_block = farmalloc(sizeof(BYTE) * DATA_SIZE);
        if(!r_th_block) {
          cputs("Not enough memory r_th_block!\n\r");
          exit(1);
        }
/*      spic = farmalloc(0xFFFF);
        if(!spic) {
          cputs("Not enough memory !\n\r");
          exit(1);
        }
*/      temp = farmalloc(16384);
        if(!temp) {
          cputs("Not enough memory temp!\n\r");
          exit(1);
        }
        dest = farmalloc(16384);
        if(!dest) {
          cputs("Not enough memory dest!\n\r");
          exit(1);
        }
        diff = farmalloc(16384);
        if(!diff) {
          cputs("Not enough memory diff!\n\r");
          exit(1);
        }
```

```
pblock = farmalloc(sizeof(struct BLOCK*) * DATA_SIZE);
if(!pblock) {
   cputs("Not enough memory pblock!\n\r");
   exit(1);
}
tb = pblock;
for(i=0;i < DATA_SIZE;i++) {
   *tb = farmalloc(sizeof(struct BLOCK));
   if(!*tb) {
      cputs("Not enough memory in pblock!\n\r");
      exit(1);
   }
   tb++;
}
sblock = farmalloc(sizeof(struct BLOCK_8BIT*) * DATA_SIZE);
if(!sblock) {
   cputs("Not enough memory sblock!\n\r");
   exit(1);
}
pb8 = sblock;
for(i=0;i < DATA_SIZE;i++) {
   *pb8 = farmalloc(sizeof(struct BLOCK_8BIT));
   if(!*pb8) {
      cputs("Not enough memory in sblock!\n\r");
      exit(1);
   }
   pb8++;
}
dct_block = farmalloc(sizeof(struct BLOCK*) * DATA_SIZE);
if(!dct_block) {
   cputs("Not enough memory dct_block!\n\r");
   exit(1);
}
tb = dct_block;
for(i=0;i < DATA_SIZE;i++) {
   *tb = farmalloc(sizeof(struct BLOCK));
   if(!*tb) {
      cputs("Not enough memory in dct_block!\n\r");
```

```
            exit(1);
        }
        tb++;
    }
    idct_block = farmalloc(sizeof(struct BLOCK*) * DATA_SIZE);
    if(!idct_block) {
        cputs("Not enough memory idct_block!\n\r");
        exit(1);
    }
    tb = idct_block;
    for(i=0;i < DATA_SIZE;i++) {
        *tb = farmalloc(sizeof(struct BLOCK));
        if(!*tb) {
            cputs("Not enough memory in idct_block!\n\r");
            exit(1);
        }
        tb++;
    }
    t_predictor = farmalloc(sizeof(struct BLOCK*) * DATA_SIZE);
    if(!t_predictor) {
        cputs("Not enough memory t_predicter!\n\r");
        exit(1);
    }
    tb = t_predictor;
    for(i=0;i < DATA_SIZE;i++) {
        *tb = farmalloc(sizeof(struct BLOCK));
        if(!*tb) {
            cputs("Not enough memory in t_predictor!\n\r");
            exit(1);
        }
        tb++;
    }
    r_predictor = farmalloc(sizeof(struct BLOCK*) * DATA_SIZE);
    if(!r_predictor) {
        cputs("Not enough memory r_predictor!\n\r");
        exit(1);
    }
    tb = r_predictor;
```

```
for(i=0;i < DATA_SIZE;i++) {
 , *tb = farmalloc(sizeof(struct BLOCK));
   if(!*tb) {
      cputs("Not enough memory in r_predictor!\n\r");
      exit(1);
   }
   tb++;
}
qblock = farmalloc(sizeof(struct BLOCK*) * DATA_SIZE);
if(!qblock) {
   cputs("Not enough memory qblock!\n\r");
   exit(1);
}
tb = qblock;
for(i=0;i < DATA_SIZE;i++) {
   *tb = farmalloc(sizeof(struct BLOCK));
   if(!*tb) {
      cputs("Not enough memory in qblock!\n\r");
      exit(1);
   }
   tb++;
}
tblock = farmalloc(sizeof(struct BLOCK*) * DATA_SIZE);
if(!tblock) {
   cputs("Not enough memory tblock!\n\r");
   exit(1);
}
tb = tblock;
for(i=0;i < DATA_SIZE;i++) {
   *tb = farmalloc(sizeof(struct BLOCK));
   if(!*tb) {
      cputs("Not enough memory in tblock!\n\r");
      exit(1);
   }
   tb++;
}
rblock = farmalloc(sizeof(struct BLOCK_8BIT*) * DATA_SIZE);
if(!rblock) {
```

```
      cputs("Not enough memory rblock!\n\r");
      exit(1);
   }
   pb8 = rblock;
   for(i=0;i < DATA_SIZE;i++) {
      *pb8 = farmalloc(sizeof(struct BLOCK_8BIT));
      if(!*pb8) {
         cprintf("Not enough memory in rblock! %d \n\r",i);
         exit(1);
      }
      pb8++;
   }
/*    rtblock = farmalloc(sizeof(struct BLOCK*) * DATA_SIZE);
   if(!rtblock) {
      cputs("Not enough memory rtblock!\n\r");
      exit(1);
   }
   tb = rtblock;
   for(i=0;i < DATA_SIZE;i++) {
      *tb = farmalloc(sizeof(struct BLOCK));
      if(!*tb) {
         cputs("Not enough memory in rtblock!\n\r");
         exit(1);
      }
      tb++;
   }*/
   clrscr();
   clearpredictor(t_predictor,r_predictor,t_th_block,r_th_block);
   count = 0;
   bits = 0L;
   xvbits = 0L;
   pname = picname;
   *pname = 0;
   snr = 0.0;
   clear_screen();
   swapbank();
   delay(DELAY_TIME);
   clear_screen();
```

```
bank = 'A';
do {
    clrscr();
    if (bits) {
       avr = (65536.0 / bits) * count;
    }
    else
       avr = 0.0;
    cputs("          1. load picture\n\r");
    cputs("          2. clear predictor\n\r");
    cputs("          3. 2-D DCT 8x8\n\r");
    cputs("          4. dpcm - coder - transmit\n\r");
    cputs("          5. receive - decoder 2-D iDCT\n\r");
    cputs("          6. save channel\n\r");
    cputs("          9. osshell\n\r");
    cputs("          0. exit\n\r");
    cputs("          C. change rate\n\r");
    cputs("          S. save picture\n\r");
    cputs("          T. change threshold\n\r");
    cputs("          Z. Zoom\n\r");
    cputs("          H. histogram\n\r");
    cputs("        now display bank ");
    highvideo();
    cprintf("%c ",bank);
    normvideo();
    cputs("space bar swapbank\n\r");
    cputs("       number of iteration ");
    highvideo();
    cprintf("%d ",count);
    normvideo();
    cputs("picture name ");
    highvideo();
    cprintf("%s\n\r",pname);
    normvideo();
    cputs("       number of bit of last frame ");
    highvideo();
    cprintf("%u %u",bit_count,xvbit_count);
    normvideo();
```

```
            cputs("time use ");
            highvideo();
            cprintf("%4.3f ",(double)bit_count / 65536.0);
            normvideo();
            cputs("sec. SNR = ");
            highvideo();
            cprintf("%5.2f ",snr);
            normvideo();
            cputs("dB.\n\r");
            cputs("      accumulate bits = ");
            highvideo();
            cprintf("%lu %lu",bits,xvbits);
            normvideo();
            cputs("average = ");
            highvideo();
            cprintf("%4.2f ",avr);
            normvideo();
            cputs("frame/sec.\n\r");
            cputs("       select item : ");
            c = getche();
            cputs("\n\r");
/*(void)getchar();*/
            switch (c) {
                case '1':loadpic(pname);
                         sub_sampling(buffer,temp,128,128);
                         pic_to_block(temp,pblock,16,16,8);
                         display(temp,128,128,128,128,0,0);
                         swapbank();
                         delay(DELAY_TIME);
                         display(temp,128,128,128,128,0,0);
                         swapbank();
                         break;
                case '2':clearpredictor(t_predictor,r_predictor,t_th_block,r_th_block);
                         count = 0;
                         bits = 0L;
                         break;
                case '3':sub_sampling(buffer,temp,128,128);
                           pic_to_block(temp,pblock,16,16,8);
```

```
                dct2_d(pblock,dct_block);
                break;
        case '4':dpcm(dct_block,t_predictor,qblock);
                threshold_encode(qblock,sblock,t_th_block);
                threshold_decode(sblock,tblock,t_th_block);
                update_tpredictor(tblock,t_predictor);
                histogram_update(sblock,hist,zero);
                bit_count = huffman_encode(sblock,channel,t_th_block);
/*                xvbit_count = sgxv_encode(sblock);*/
                bits += bit_count;
                xvbits += xvbit_count;
                huffman_decode(channel,rblock,r_th_block);
                break;
        case '5':threshold_decode(rblock,tblock,r_th_block);
                idpcm(tblock,r_predictor);
                update_rpredictor(tblock,r_predictor);
                idct2_d(tblock,idct_block);
                if(processed)
                  display(dest,128,128,128,128,128,128);
                block_to_pic(idct_block,dest,16,16,8);
                display(dest,128,128,128,128,128,0);
                  snr = SNR(temp,dest,diff,128,128,128);
                display(diff,128,128,128,128,0,128);
                swapbank();
                count++;
                processed = TRUE;
                break;
/*              case '6':save_channel(channel);
                break;
*/              case '9':system("\command");
                break;
/*              case 's':save_pic();
                break;
        case 'z':
        case 'Z':expand(spic);
                break;
        case 'c':
        case 'C':changerate();
```

```
                    break;
                case 't':
                case 'T':changethreshold();
                    break;
                case 'h':
                case 'H':histogram(dct_block);
                    break;
*/              case ' ':swapbank();
                    if(bank == 'A')
                        bank = 'B';
                    else
                        bank = 'A';
                    break;
            }
        }
        while(c != '0');
        histogram_save(hist,zero);
        farfree(r_th_block);
        farfree(t_th_block);
        farfree(temp);
/*      farfree(spic);*/
        farfree(qblock);
        farfree(sblock);
        farfree(tblock);
        farfree(t_predictor);
        farfree(r_predictor);
        farfree(pblock);
        farfree(channel);
        farfree(rblock);
/*      farfree(rtblock);*/
        farfree(dct_block);
        farfree(idct_block);
        farfree(dest);
        farfree(diff);
}
/*
void save_pic()
{
```

```
        char s[35],c;

        cputs("Enter filename to save : ");
        gets(s);
        c = 'Y';
        if(exist(s) == 0) {
          cputs("Overwrite (y/n)");
          c = getche();
        }
        if((c == 'y') || (c == 'Y')) {
          swapbank();
          delay(100);
          buffer_to_file(s);
          swapbank();
        }
}
void save_channel(BYTE *ch)
{
        BYTE data,code;
        int      i,bitcount;
        char s[40];
        FILE *fp;

        cputs("Enter file name to save : ");
        gets(s);
        fp = fopen(s,"wb");
        for(i=0;i < 8192;i++) {
              code = 0;
              for(bitcount =0;bitcount < 8;bitcount++) {
                 code <<= 1;
                 data = *ch++;
                 code |= data;
              }
              fputc(code,fp);
        }
        fclose(fp);
}
*/
```

```
void histogram_save(unsigned long *hist,unsigned long *zero)
{
      int i;
      FILE *fp1,*fp2;
      char s[35];

      cputs("Enter filename to save  histogram : ");
      gets(s);
      fp1 = fopen(s,"wt");
      if(!fp1) {
         cprintf("\007Can't open file %s \n\r",s);
         return;
      }
      for(i=0;i < 33;i++) {
         fprintf(fp1,"%d %ld\n",i,*hist++);
      }
      fclose(fp1);
      cputs("Enter filename to save zero count : ");
      gets(s);
      fp2 = fopen(s,"wt");
      if(!fp2) {
         cprintf("\007Can't open file %s \n\r",s);
         return;
      }
      for(i=0;i < 64;i++) {
         fprintf(fp2,"%d %ld\n",i,*zero++);
      }
      fclose(fp2);
}
void histogram_update(struct BLOCK_8BIT **pb,unsigned long *hist,unsigned long *zero)
{
      int j,i,data,zero_count;

      for(i=0;i < DATA_SIZE;i++) {
         j = 0;
         zero_count = 0;
         do {
            data = (*pb)->pixel[j];
```

```
            if(data > 0) {
               ++hist[data];
               j++;
            }
            else if(data < 0) {
               ++hist[-data];
               j++;
            }
            else { /* data = 0 */
                 do {
                    zero_count++;
                    j++;
                 }
                 while(((*pb)->pixel[j] == 0) && (j < BLOCK_SIZE));
                 if(j < BLOCK_SIZE) {
                     ++zero[zero_count];
                 }
                 zero_count = 0;
            }
         }
         while(j < BLOCK_SIZE);
         pb++;
      }
}
unsigned huffman_encode(struct BLOCK_8BIT **pb,BYTE *ch,BYTE *th)
{
      int k,j,i,temp[BLOCK_SIZE],data;
      unsigned bitcount;
      BYTE nbit,hcode,zero_count,bdata;

      bitcount = 0;
      for(i=0;i < DATA_SIZE;i++) {
         j = 0;
         zero_count = 0;
         if(*th == TH_HIGH) {
           *ch++ = 1;
         }
         else {
```

```
            *ch++ = 0;
          }
          bitcount++;
          do {
             data = (*pb)->pixel[j];
             if(data > 0) {
                if(data < 13) {
                  nbit = huffman[data].bit;
                  hcode = huffman[data].code;
                  for(k=nbit-2;k >= 0;k--) {
                     *ch++ = (hcode >> k) & 1;
                  }
                  *ch++ = 0;  /* sign bit */
                }
                else {    /* data >= 13 */
                  nbit = huffman[13].bit;
                  hcode = huffman[13].code;
                  for(k=nbit-2;k >= 0;k--) {
                     *ch++ = (hcode >> k) & 1;
                  }
/*                  *ch++ = 0;*/
                  hcode = (BYTE)data;
                  for(k=7;k >=0 ;k--) {
                     *ch++ = (hcode >> k) & 1;
                  }
/*                  nbit += 8;*/
                  nbit += 6; /* new */
                }
                j++;
                bitcount += nbit;
             }
             else if(data < 0) {
                if(data > -13) {
                  nbit = huffman[-data].bit;
                  hcode = huffman[-data].code;
                  for(k=nbit-2;k >= 0;k--) {
                     *ch++ = (hcode >> k) & 1;
                  }
```

```
                *ch++ = 1;  /* sign bit - */
              }
              else { /* data <= -13 */
                nbit = huffman[13].bit;
                hcode = huffman[13].code;
                for(k=nbit-2;k >= 0;k--) {
                   *ch++ = (hcode >> k) & 1;
                }
/*                *ch++ = 1;*/
/*                hcode = (BYTE)(-data);*/
                hcode = data;
                for(k=7;k >=0 ;k--) {
                   *ch++ = (hcode >> k) & 1;
                }
/*                nbit += 8;*/
                nbit += 6; /* new */
              }
              j++;
              bitcount += nbit;
            }
            else { /* data = 0 */
                do {
                   zero_count++;
                   j++;
                }
                while(((*pb)->pixel[j] == 0) && (j < BLOCK_SIZE));
                if(j < BLOCK_SIZE) {
                  nbit = 3;     /* run length prefix */
                  bitcount += nbit;
                  hcode = 2;
                  for(k=nbit-1;k >= 0;k--) {
                     *ch++ = (hcode >> k) & 1;
                  }
                  if(zero_count < 30) {
                     nbit =  run_length[zero_count].bit;
                     hcode = run_length[zero_count].code;
                     for(k=nbit-1;k >= 0;k--) {
                        *ch++ = (hcode >> k) & 1;
```

```
                    }
                     bitcount += nbit;
                  }
                  else {
                    nbit = run_length[30].bit;
                    hcode = run_length[30].code;
                    hcode <<= 6;
                    hcode |= zero_count;
                    for(k=nbit-1;k >= 0;k--) {
                       *ch++ = (hcode >> k) & 1;
                    }
                     bitcount += nbit;
                  }
               }
               zero_count = 0;
            }
         }
         while(j < BLOCK_SIZE);
         nbit = 4; /* EOB */
         hcode = 1;
         for(k=nbit-1;k >= 0;k--) {
            *ch++ = (hcode >> k) & 1;
         }
         pb++;
         th++;
         bitcount += nbit;
      }
      return(bitcount);
}
/*
unsigned sgxv_encode(struct BLOCK_8BIT **pb)
{
      int k,j,i,temp[BLOCK_SIZE],data;
      unsigned bitcount;
      BYTE nbit,hcode,run,bdata;

      bitcount = 0;
      for(i=0;i < DATA_SIZE;i++) {
```

```
        j = 0;
        run = 0;
        bitcount++;
        do {
           data = (*pb)->pixel[j];
           data = abs(data);
           if(data != 0) {
              if(run > 26) {
                 nbit = 18;
              }
              else if(data > 15) {
                 nbit = 18;
              }
              else {
                 nbit = vlc[run][data];
                 if(nbit == 20)
                   nbit = 18;
              }
              j++;
              bitcount += nbit;
              run = 0;
           }
           else { /* data = 0 */
                run = 0;
                do {
                   run++;
                   j++;
                }
                while(((*pb)->pixel[j] == 0) && (j < BLOCK_SIZE));
           }
        }
        while(j < BLOCK_SIZE);
        nbit = 2; /* EOB */
        pb++;
        bitcount += nbit;
     }
     return(bitcount);
}
```

```
*/
void huffman_decode(BYTE *ch,struct BLOCK_8BIT **pb,BYTE *th)
{
      int k,j,i,data,decoded,sign,coeff_count,block_count,eob;
      BYTE bdata;

      block_count = 0;
      do {
         coeff_count = 0;
         eob = FALSE;
         if(*ch++) {
           *th = TH_HIGH;
         }
         else {
           *th = TH_LOW;
         }
         do {
            data = *ch++;
            if(data) /* 1 */
              decoded = 1;
            else { /* data = 0 */
              data = *ch++;
              if(data) {    /* 01 */
                data = *ch++;
                if(data) {     /* 011 */
                  data = *ch++;
                  if(data) {       /* 0111 */
                    decoded = 3;
                  }
                  else {          /* 0110 */
                    data = *ch++;
                    if(data) {     /* 01101 */
                      decoded = 5;
                    }
                    else {       /* 01100 */
                      data = *ch++;
                      if(data) {     /* 011001 */
                        decoded = 6;
```

```
        }
        else {   /* 011000 */
          data = *ch++;
          if(data) {    /* 0110001 */
            decoded = 8;
          }
          else {        /* 0110000 */
            data = *ch++;
            if(data) {    /* 01100001 */
              decoded = 12;
            }
            else          /* 01100000 */
              decoded = 10;
          }
        }
      }
    }
  }
  else {  /* 010  run length */
      decoded = 0;
  }
}
else { /* 00 */
    data = *ch++;
    if(data) {  /* 001 */
      decoded = 2;
    }
    else {  /* 000 */
      data = *ch++;
      if(data) { /* 0001  EOB */
        decoded = 255;
      }
      else { /* 0000 */
        data = *ch++;
        if(data) { /* 00001 */
          decoded = 4;
        }
        else { /* 00000 */
```

```
                data = *ch++;
                if(data) { /* 000001 */
                  decoded = 13;
                }
                else { /* 000000 */
                  data = *ch++;
                  if(data) { /* 0000001 */
                    decoded = 7;
                  }
                  else { /* 0000000 */
                    data = *ch++;
                    if(data) { /* 00000001 */
                      decoded = 11;
                    }
                    else { /* 00000000 */
                      decoded = 9;
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  switch(decoded) {
      case  0:data = *ch++;
                if(data) { /* 1 */
                  data = *ch++;
                  if(data) {  /* 11 */
                    decoded = 1;
                  }
                  else {      /* 10 */
                    data = *ch++;
                    if(data) {  /* 101 */
                      decoded = 2;
                    }
                    else {      /* 100 */
                      data = *ch++;
```

```
            if(data) {   /* 1001 */
              data = *ch++;
              if(data) {    /* 10011 */
                decoded = 10;
              }
              else {        /* 10010 */
                decoded = 7;
              }
            }
            else {       /* 1000 */
              data = *ch++;
              if(data) {    /* 10001 */
                decoded = 9;
              }
              else {        /* 10000 */
                data = *ch++;
                if(data) {     /* 100001 */
                  decoded = 15;
                }
                else {         /* 100000 */
                  decoded = 12;
                }
              }
            }
          }
        }
      }
      else {     /* 0 */
        data = *ch++;
        if(data) {   /* 01 */
          data = *ch++;
          if(data) {    /* 011 */
            decoded = 3;
          }
          else {        /* 010 */
            data = *ch++;
            if(data) {     /* 0101 */
              decoded = 4;
```

```
    }
    else {          /* 0100 */
      data = *ch++;
      if(data) {        /* 01001 */
        decoded = 8;
      }
      else {            /* 01000 */
        decoded = 6;
      }
    }
  }
}
else {        /* 00 */
  data = *ch++;
  if(data) {    /* 001 */
    data = *ch++;
    if(data) {      /* 0011 */
      decoded = 5;
    }
    else {          /* 0010 */
      data = *ch++;
      if(data) {        /* 00101 */
        data = *ch++;
        if(data) {          /* 001011 */
          decoded = 17;
        }
        else {              /* 001010 */
          decoded = 13;
        }
      }
      else {            /* 00100 */
        data = *ch++;
        if(data) {          /* 001001 */
          decoded = 14;
        }
        else {              /* 001000 */
          decoded = 11;
        }
```

```
    }
  }
}
else {        /* 000 */
  data = *ch++;
  if(data) {     /* 0001 */
    data = *ch++;
    if(data) {      /* 00011 */
      data = *ch++;
      if(data) {       /* 000111 */
        data = *ch++;
        if(data) {        /* 0001111 */
          decoded = 25;
        }
        else {            /* 0001110 */
          decoded = 21;
        }
      }
      else {           /* 000110 */
        data = *ch++;
        if(data) {        /* 0001101 */
          data = *ch++;
          if(data) {         /* 00011011 */
            decoded = 29;
          }
          else {             /* 00011010 */
            decoded = 27;
          }
        }
        else {           /* 0001100 */
          data = *ch++;
          if(data) {         /* 00011001 */
            decoded = 28;
          }
          else {             /* 00011000 */
            decoded = 26;
          }
        }
```

```
    }
  }
  else {          /* 00010 */
    decoded = 30;
  }
}
else {          /* 0000 */
  data = *ch++;
  if(data) {      /* 00001 */
    data = *ch++;
    if(data) {      /* 000011 */
      decoded = 16;
    }
    else {          /* 000010 */
      data = *ch++;
      if(data) {      /* 0000101 */
        decoded = 23;
      }
      else {          /* 0000100 */
        decoded = 19;
      }
    }
  }
  else {          /* 00000 */
    data = *ch++;
    if(data) {      /* 000001 */
      data = *ch++;
      if(data) {      /* 0000011 */
        decoded = 24;
      }
      else {          /* 0000010 */
        decoded = 20;
      }
    }
    else {          /* 000000 */
      data = *ch++;
      if(data) {      /* 0000001 */
        decoded = 22;
```

```
                    }
                    else {              /* 0000000 */
                      decoded = 18;
                    }
                  }
                }
              }
            }
          }
          if(decoded == 30) {
            bdata = 0;
            for(i=0;i < 5;i++) {
               bdata += *ch++;
               bdata <<= 1;
            }
            bdata += *ch++;
            decoded = bdata;
          }
          for(i=0;i < decoded;i++) {
             if(coeff_count >= BLOCK_SIZE) {
                cputs("\n\007 zigzag overflow in run length\n");
                cprintf("coeff = %d decoded = %d\n",coeff_count,decoded);
                getch();
                return;
             }
             (*pb)->pixel[coeff_count++] = 0;
          }
          break;
case  13:/*sign = *ch++;*/
          bdata = 0;
          for(i=0;i < 7;i++) {
             bdata |= *ch++;
             bdata <<= 1;
          }
          bdata |= *ch++;
          decoded = (char)bdata;
/*          if(sign)
```

```
                    decoded = -bdata;
                  else
                    decoded = bdata;*/
                     if(coeff_count >= BLOCK_SIZE) {
                        cputs("\n\007 zigzag overflow in amplitude\n");
                        cprintf("coeff = %d\n",coeff_count);
                        getch();
                        return;
                     }
                       (*pb)->pixel[coeff_count++] = decoded;
                  break;
          case 255:eob = TRUE;
                  break;
          default :sign = *ch++;
                  if(sign)
                    decoded *= -1;
                     if(coeff_count >= BLOCK_SIZE) {
                        cputs("\n\007 zigzag overflow in default\n");
                        cprintf("coeff = %d\n",coeff_count);
                        getch();
                        return;
                     }
                       (*pb)->pixel[coeff_count++] = decoded;
                  break;
    }
cprintf("%3d %2d     \r",block_count,coeff_count);
}
while((!eob) && (coeff_count < BLOCK_SIZE));
block_count++;
if(eob) {
  for(;coeff_count < BLOCK_SIZE;) {
     if(coeff_count >= BLOCK_SIZE) {
         cputs("\n\007 zigzag overflow in EOB\n");
         cprintf("coeff = %d\n",coeff_count);
         getch();
         return;
       }
     (*pb)->pixel[coeff_count++] = 0;
```

```
            }
          }
          else {
            data = *ch++;
            if(data) { /* 1 */
              cputs("\n\007 sync error\n");
              return;
            }
            else {     /* 0 */
              data = *ch++;
              if(data) {  /* 01 */
                cputs("\n\007 sync error\n");
                return;
              }
              else {      /* 00 */
                data = *ch++;
                if(data) {  /* 001 */
                  cputs("\n\007 sync error\n");
                  return;
                }
                else {      /* 000 */
                  data = *ch++;
                  if(data) {  /* 0001 */
                  }
                  else {      /* 0000 */
                    cputs("\n\007 sync error\n");
                    getch();
                    return;
                  }
                }
              }
            }
          }
          pb++;
          th++;
        }
        while(block_count < DATA_SIZE);
}
```

```
/*
void changethreshold(void)
{
      char s[5];

      cprintf("threshold now = %d\n",thset);
      cputs("Enter new threshold : ");
      gets(s);
      thset = atoi(s);
}
void changerate(void)
{
      char s[5];

      cprintf("rate now = %d frame / sec.\n",DATA_SIZE / bound);
      cputs("Enter frame rate require : ");
      gets(s);
      bound = DATA_SIZE / atoi(s);
}
void histogram(struct BLOCK **pb)
{
      int i,j,data;
      FILE *fp;
      long h[256],*hp;
      char s[35];

      setmem(h,sizeof(long) * 256,0);
      for(i=0;i < DATA_SIZE;i++) {
         for(j=0;j < BLOCK_SIZE;j++) {
            data = abs((*pb)->pixel[j]);
            if(data < 256) {
               ++h[data];
            }
         }
         pb++;
      }
      cputs("Enter filename to save : ");
      gets(s);
```

```
        fp = fopen(s,"wt");
        if(!fp) {
           cprintf("\007Can't open file %s \n",s);
           return;
        }
        hp = h;
        for(i=0;i < 256;i++) {
           fprintf(fp,"%d %ld\n",i,*hp++);
        }
        fclose(fp);
}

unsigned bitcount(struct BLOCK **dest,BYTE *th)
{
        unsigned bit;
        int i,j,data,temp[64],zero_count,block_bit;

        bit = 0;
        for(i=0;i < DATA_SIZE;i++) {
           /* zigzag scan */
           for(j=0;j < BLOCK_SIZE;j++) {
              temp[j] = (*dest)->pixel[zigzag[j]];
           }

           zero_count = 0;
           j = 0;
           block_bit = 0;
           do {
              data = abs(temp[j]);
              if(data != 0) {
                 if(data >= 13)
                    block_bit += 15;
                 else
                    block_bit += huffman[data].bit;
                 j++;
              }
              else {
                 do {
```

```
                zero_count++;
                j++;
            }
            while ((temp[j] == 0) && (j < BLOCK_SIZE));
            if(j != BLOCK_SIZE) {
                block_bit += 3; /* run length prefix */
                if(zero_count > 30)
                  block_bit += 11;
                else
                  block_bit += run_length[zero_count].bit;
            }
            zero_count = 0;
          }
        }
        while (j < BLOCK_SIZE);
        block_bit += 4; /* EOB */
        bit += block_bit;
        if(block_bit > bound) {
          *th += 4;
        }
        else if((block_bit < bound) && (*th > 4)) {
          *th -= 4;
        }
        dest++;
        th++;
    }
/*      cprintf("data use %u bits\n",bit);
    cputs("....press any key....\n");
    getch();
*/      return(bit);
}
*/
void copyblock(struct BLOCK **source,struct BLOCK **dest)
{
    int i,j;

    for(i=0;i < DATA_SIZE;i++) {
      for(j=0;j < BLOCK_SIZE;j++) {
```

```
            (*dest)->pixel[j] = (*source)->pixel[j];
         }
         source++;
         dest++;
      }
}
void dpcm(struct BLOCK **source,struct BLOCK **pred,struct BLOCK **dest)
{
      int i,j,data;

      for(i=0;i < DATA_SIZE;i++) {
         for(j=0;j < BLOCK_SIZE;j++) {
            data = (*source)->pixel[j] - (*pred)->pixel[j];
            (*dest)->pixel[j] = data;
         }
         source++;
         pred++;
         dest++;
      }
}
char quantize(int data)
{
      if(data < 0) {
         if(data < MINLEVEL/*-1024*/) {
            data = MINLEVEL/*-1024*/;
         }
      }
      else {
         if(data > MAXLEVEL/*1023*/) {
            data = MAXLEVEL/*1023*/;
         }
      }
      return(data);
}
void update_tpredictor(struct BLOCK **qb,struct BLOCK **pred)
{
      int i,j;
```

```
        for(i=0;i < DATA_SIZE;i++) {
            for(j=0;j < BLOCK_SIZE;j++) {
                (*pred)->pixel[j] += (*qb)->pixel[j];
            }
            pred++;
            qb++;
        }
}
void threshold_encode(struct BLOCK **source,struct BLOCK_8BIT **dest,BYTE *th)
{
        int i,j,data;
        char s[10];

/*      cputs("Enter threshold : ");
        gets(s);
        threshold = atoi(s);
        cputs("Enter stepsize : ");
        gets(s);
        stepsize = atoi(s);
*/      for(i=0;i < DATA_SIZE;i++) {
          if(abs((*source)->pixel[0]) >= thset)
            *th = TH_HIGH;
          else
            *th = TH_LOW;
          for(j=0;j < BLOCK_SIZE;j++) {
            data = (*source)->pixel[j];
            if(data < 0) {
              if(data >= -(*th))
                (*dest)->pixel[j] = 0;
              else {
                (*dest)->pixel[j] = quantize((data + *th) / *th);
                }
            }
            else {
              if(data <= *th)
                (*dest)->pixel[j] = 0;
              else {
                (*dest)->pixel[j] = quantize((data - *th) / *th);
```

```
                }
            }
        }
        source++;
        dest++;
        th++;
    }
}
void threshold_decode(struct BLOCK_8BIT **source,struct BLOCK **dest,BYTE *th)
{
    int i,j,data;
    unsigned count;

    count = 0;
    for(i=0;i < DATA_SIZE;i++) {
        for(j=0;j < BLOCK_SIZE;j++) {
            data = (*source)->pixel[j];
            if(data > 0) {
              (*dest)->pixel[j] = (data * (*th)) + *th;
              ++count;
            }
            else if(data < 0) {
              (*dest)->pixel[j] = (data * (*th)) - *th;
              ++count;
            }
            else if(data == 0) {
              (*dest)->pixel[j] = 0;
            }
        }
        source++;
        dest++;
        th++;
    }
/*      cprintf("coefficience remain %d\n",count);
    cputs("......press any key.......");
    getch();*/
}
void idpcm(struct BLOCK **source,struct BLOCK **pred)
```

```
{
        int i,j;

        for(i=0;i < DATA_SIZE;i++) {
           for(j=0;j < BLOCK_SIZE;j++) {
              (*source)->pixel[j] += (*pred)->pixel[j];
           }
           source++;
           pred++;
        }
}
void update_rpredictor(struct BLOCK **qb,struct BLOCK **pred)
{
        int i,j;

        for(i=0;i < DATA_SIZE;i++) {
           for(j=0;j < BLOCK_SIZE;j++) {
              (*pred)->pixel[j] = (*qb)->pixel[j];
           }
           pred++;
           qb++;
        }
}
void clearpredictor(struct BLOCK **t_pred,struct BLOCK **r_pred,BYTE *tblock,BYTE *rblock)
{
        int i,j;

        for(i=0;i < DATA_SIZE;i++) {
           (*t_pred)->pixel[0] = DC_INIT;
           (*r_pred)->pixel[0] = DC_INIT;
           *tblock++ = thset;
           *rblock++ = thset;
           for(j=1;j < BLOCK_SIZE;j++) {
              (*t_pred)->pixel[j] = 0;
              (*r_pred)->pixel[j] = 0;
           }
           t_pred++;
           r_pred++;
```

```
        }
}

void dct2_d(struct BLOCK **source,struct BLOCK **dest)
{
        int i,j;

        for(i=0;i < DATA_SIZE;i++) {
/*              dct8x8(*source);*/
            fdct8x8(*source);
            for(j=0;j < BLOCK_SIZE;j++) {
                (*dest)->pixel[j] = (*source)->pixel[zigzag[j]] / 4;
            }
            source++;
            dest++;
            cprintf("%4d\r",i);
        }
        cputs("\n\r");
}
void idct2_d(struct BLOCK **source,struct BLOCK **dest)
{
        int i,j;

        zigzag_scan(source,DATA_SIZE);
        for(i=0;i < DATA_SIZE;i++) {
/*              idct8x8(*source);*/
            ifdct8x8(*source);
            for(j=0;j < BLOCK_SIZE;j++) {
                (*dest)->pixel[j] = (*source)->pixel[j] / 4;
            }
            source++;
            dest++;
            cprintf("%4d\r",i);
        }
        cputs("\n\r");
}

void loadpic(char *s)
```

```
{
      int result;

      cprintf("Enter filename  : ");
      (void)gets(s);
      result = file_to_buffer(s);
      if(result) {
        cputs("\007Read error or file not found !\n");
      }
}
/*
void conv_to_pic(struct BLOCK **pb)
{
      int i,j,m;
      BYTE *source,*sp,*lp,*bp,*ptr;

      source = MK_FP(0xA000,0);
      setmem(source,0xFFFF,0);
      ptr = lp = bp = sp = source;
      for(m=0;m < DATA_SIZE;m++) {
         if(((m / 32) > 0) && (!(m % 32))) {
           sp += 2048;
           bp = sp;
         }
         lp = bp;
         for(j=0;j<8;j++) {
            ptr = lp;
            for(i=0;i<8;i++) {
               *ptr++ = (unsigned char)abs((*pb)->pixel[8*j+i]);
            }
            lp += 256;
         }
         pb++;
         bp += 8;
      }
}

void expand(BYTE *temp)
```

```
{
        int x,y;
        char s[5];

        cputs("Enter coordinate x(0-127) : ");
        gets(s);
        x = atoi(s);
        cputs("Enter coordinate y(0-127) : ");
        gets(s);
        y = atoi(s);
        zoom(x,y,128,128,0,0);
}
*/
void conv_to_smallpic(struct BLOCK **pb)
{
        int i,j,m;
        BYTE *source,*sp,*lp,*bp,*ptr;

        source = MK_FP(0xA000,0);
        setmem(source,0xFFFF,0);
        ptr = lp = bp = sp = source;
        for(m=0;m < DATA_SIZE;m++) {
           if(((m / 16) > 0) && (!(m % 16))) {
             sp += 2048;
             bp = sp;
           }
           lp = bp;
           for(j=0;j<8;j++) {
              ptr = lp;
              for(i=0;i<8;i++) {
                 *ptr++ = (unsigned char)abs((*pb)->pixel[8*j+i]);
              }
              lp += 256;
           }
           pb++;
           bp += 8;
        }
}
```

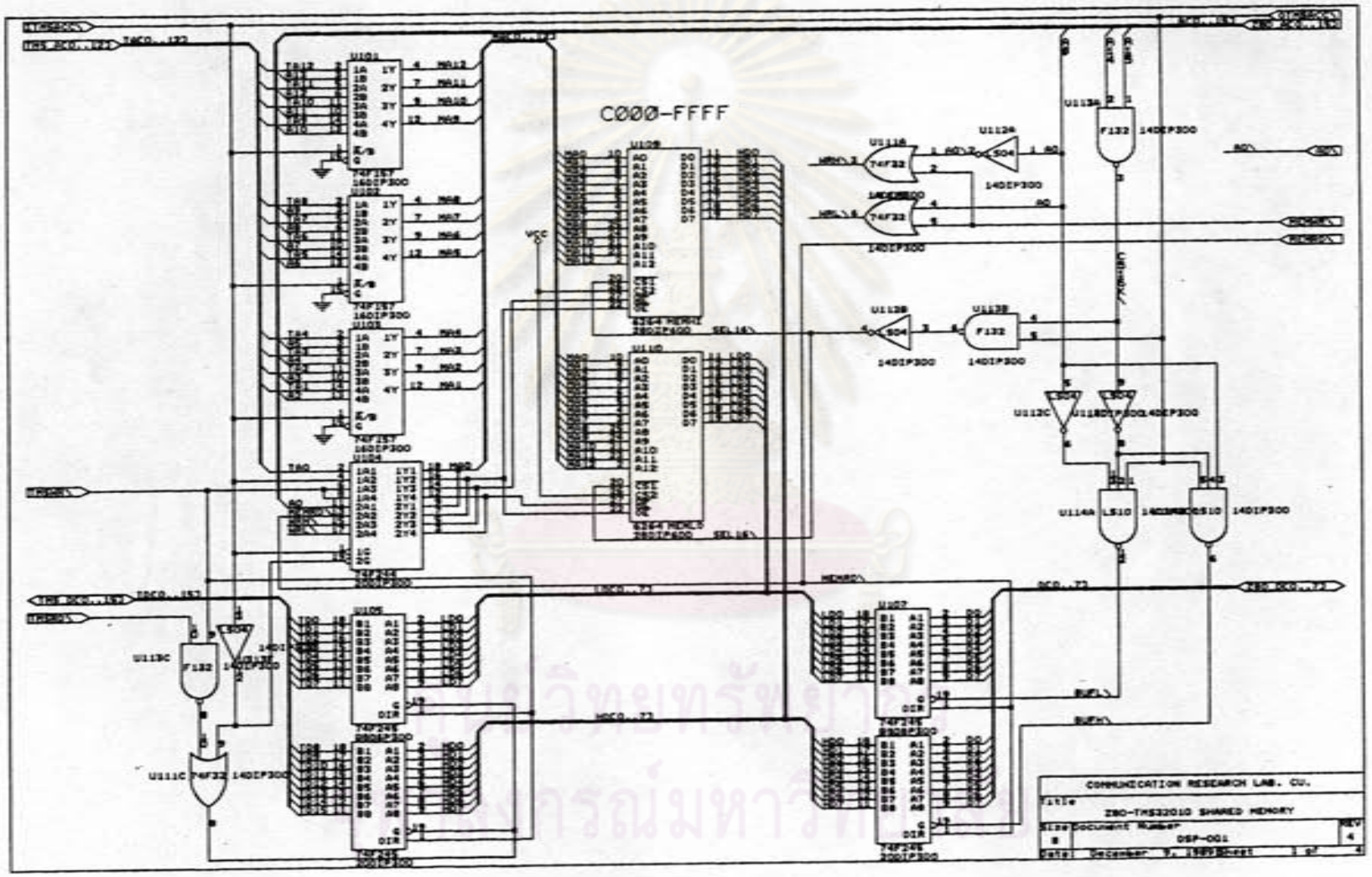
ภาคผนวก ข

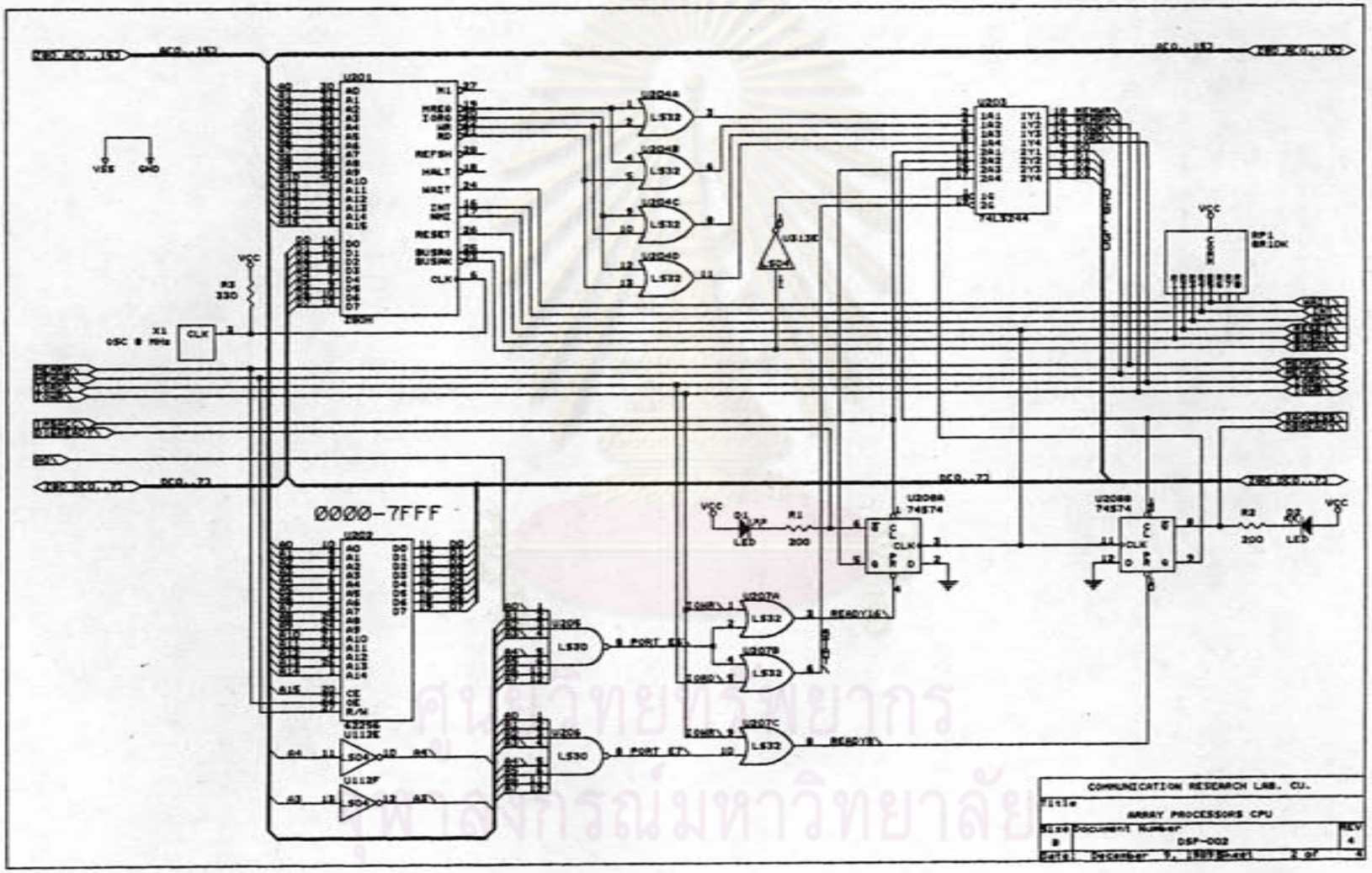
วงจรของระบบที่ได้สร้างขึ้น

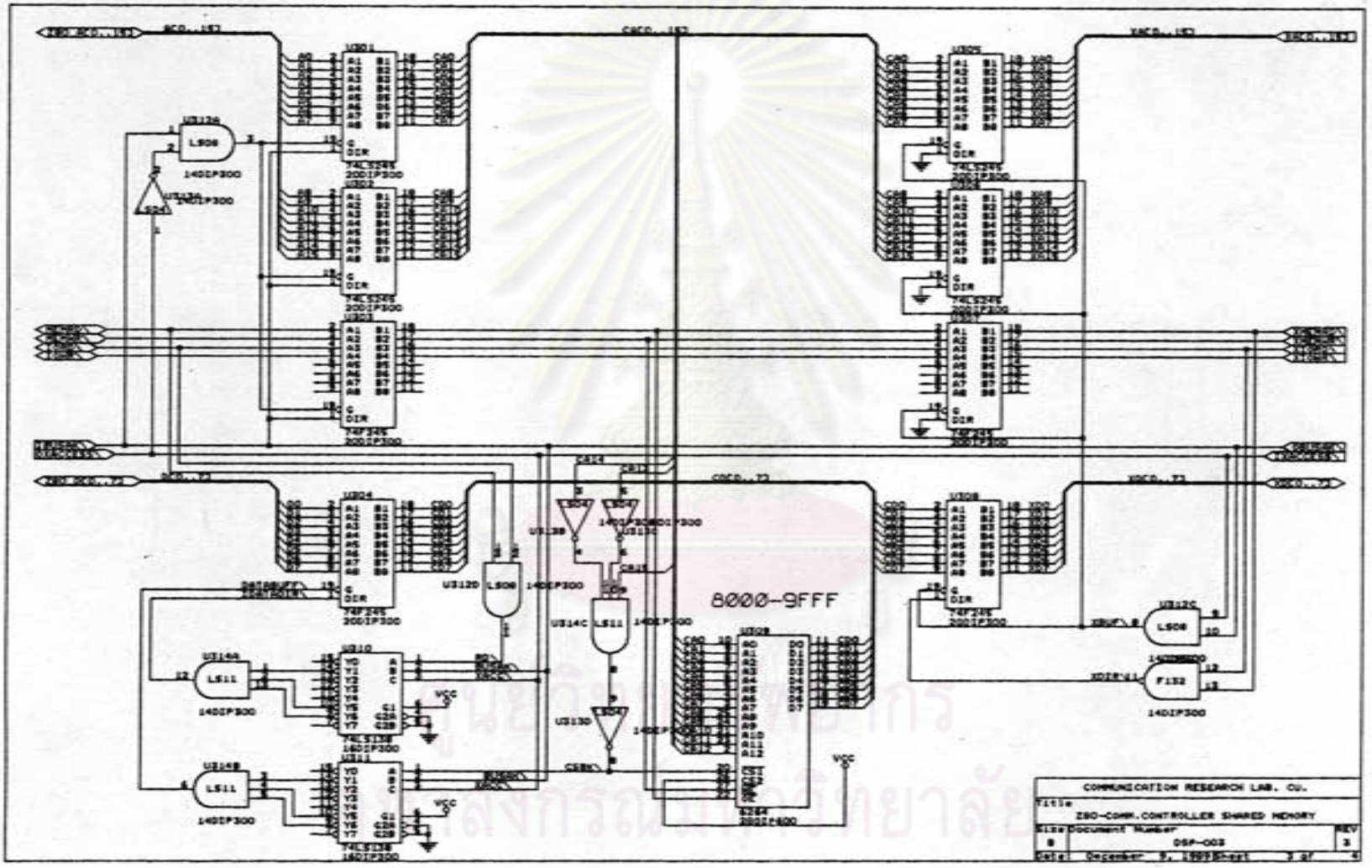
ภาคผนวกนี้แสดงวงจรต่างๆของระบบที่ได้สร้างขึ้นโดยมีรายละเอียดดังนี้

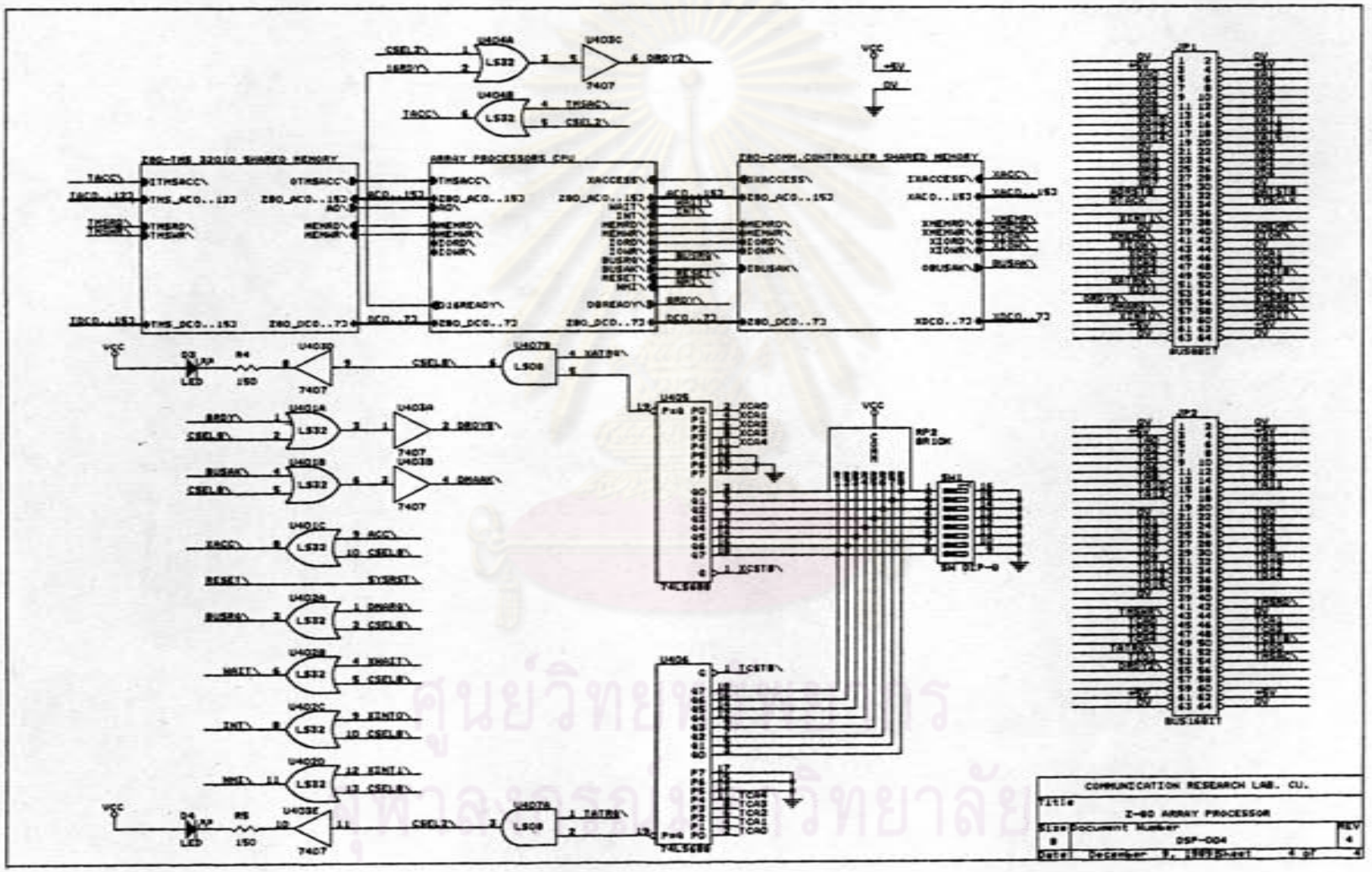
Document Number	รายละเอียด
DSP-001	วงจรของไมโครโปรเซสเซอร์การ์ด ในส่วน DATA MEMORY 16 BIT
DSP-002	วงจรของไมโครโปรเซสเซอร์การ์ด ส่วนควบคุม
DSP-003	วงจรของไมโครโปรเซสเซอร์การ์ด ในส่วน DATA MEMORY 8 BIT
DSP-004	วงจรของไมโครโปรเซสเซอร์การ์ด ในส่วนเชื่อมต่อกับบัส
DSP-005	วงจรของส่วนเชื่อมต่อกับ IBM PC
DSP-006	วงจรของมาสเตอร์การ์ด
DSP-007	วงจรของ DSP การ์ด ส่วนสร้างสัญญาณแอดเดรสของข้อมูลด้าน 16 บิต
DSP-007.2	วงจรของ DSP การ์ด ส่วนสร้างสัญญาณแอดเดรสของข้อมูลด้าน 8 บิต
DSP-007.3	วงจรของ DSP การ์ด ส่วน PROGRAM MEMORY ด้าน 8 บิตล่าง
DSP-007.4	วงจรของ DSP การ์ด ส่วน PROGRAM MEMORY ด้าน 8 บิตบน
DSP-007.5	วงจรของ DSP การ์ด ส่วนสลับแอดเดรส
DSP-008	วงจรของ DSP การ์ด ส่วน CPU
DSP-009	วงจรของ DSP การ์ด ส่วนติดต่อกับ VIDEO MEMORY
DSP-010	วงจรของ DSP การ์ด ส่วนเชื่อมต่อกับบัส

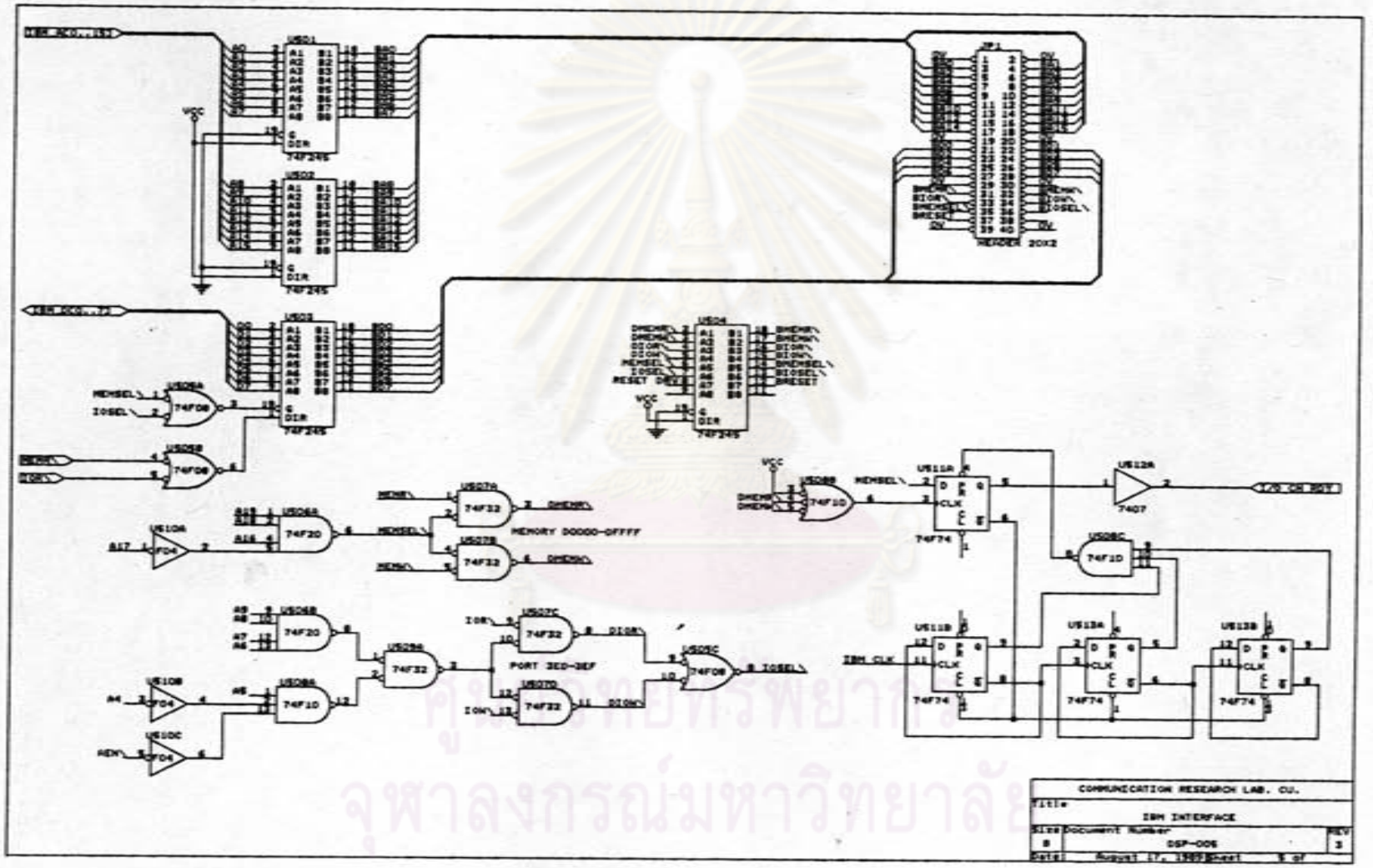
วงจรต่างๆเหล่านี้เขียนโดยใช้โปรแกรม Orcad SDT III

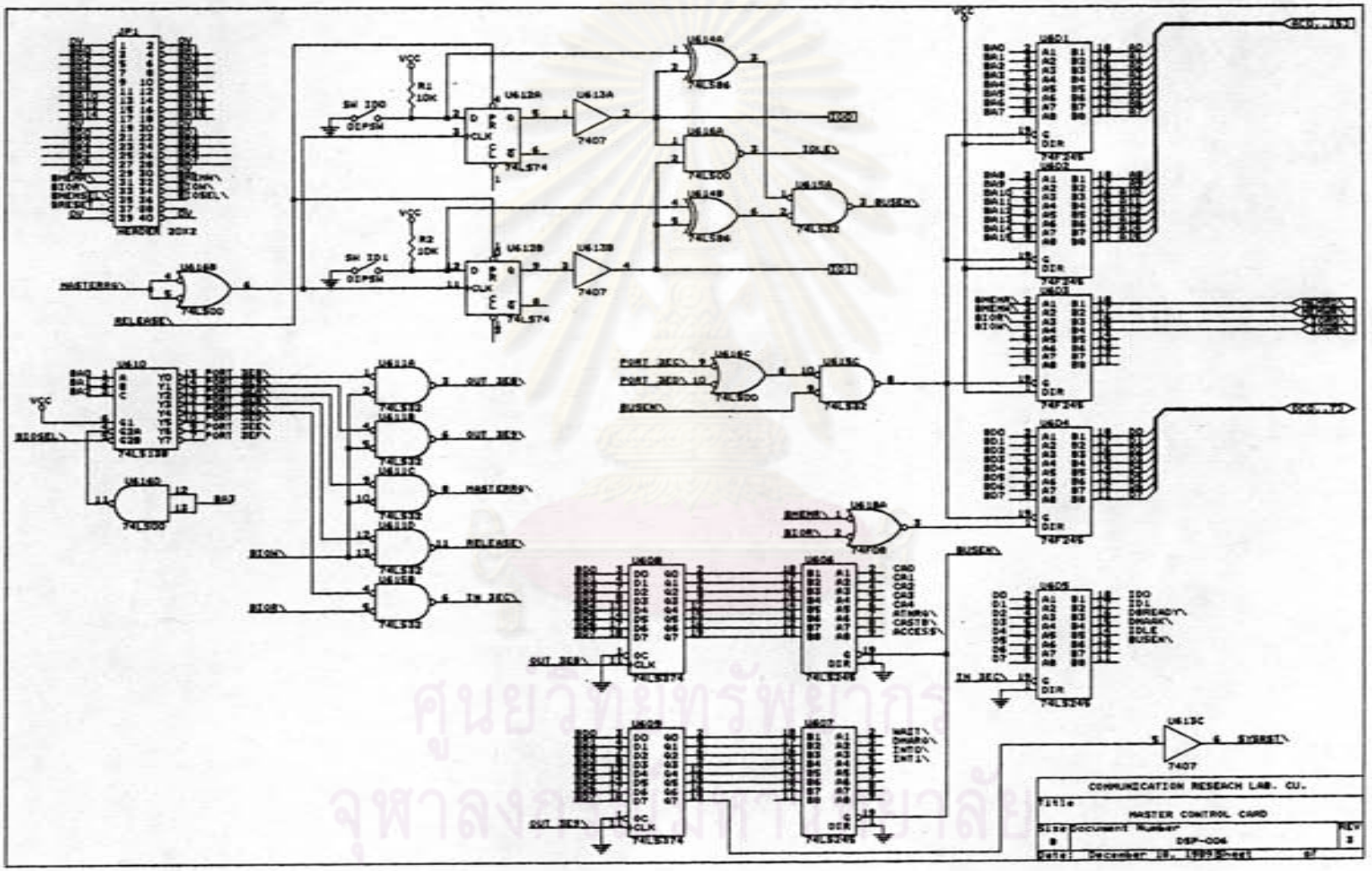

C000-FFFF
COMMUNICATION RESEARCH LAB. CU.
Z80-TMS32010 SHARED MEMORY
DSP-001

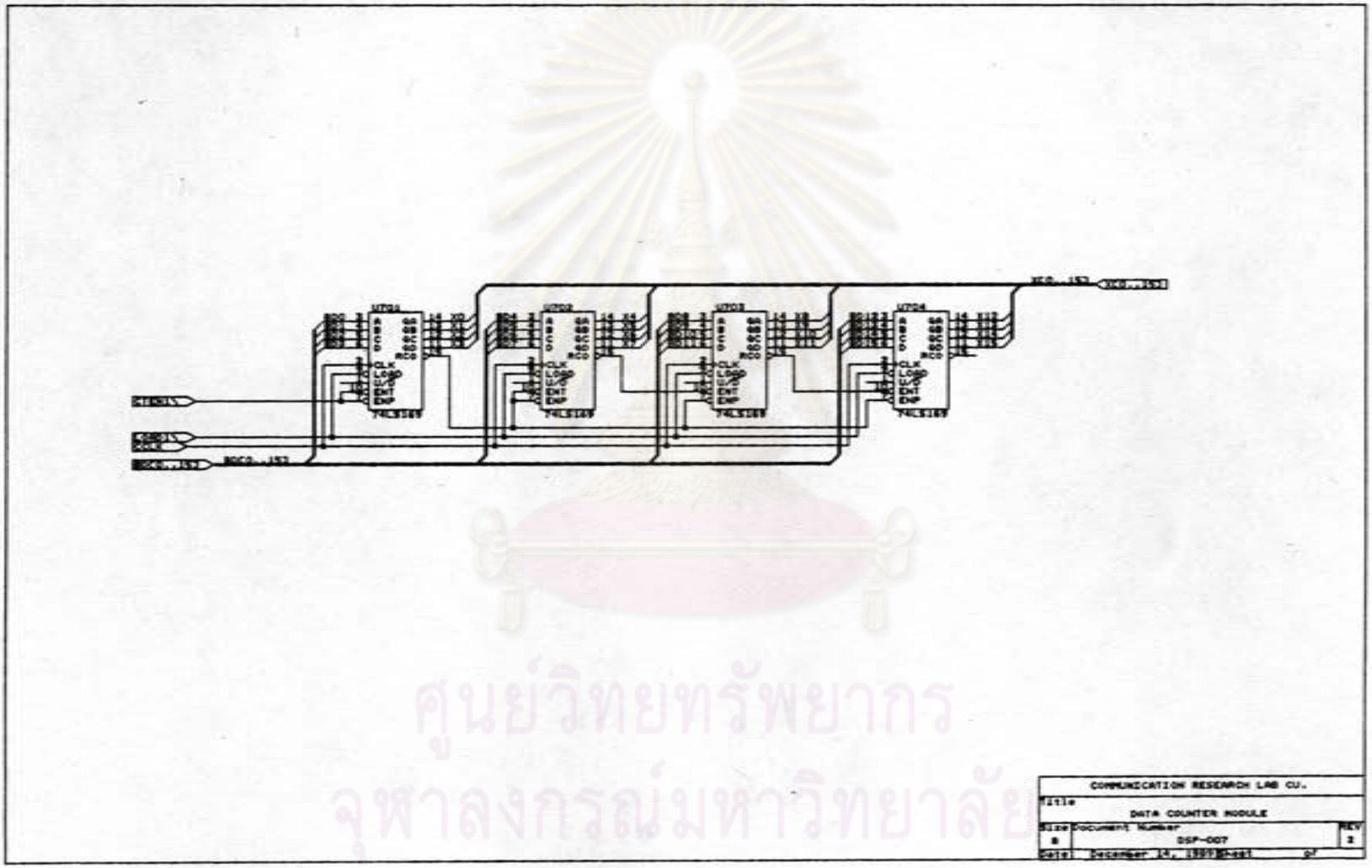

0000-7FFF
COMMUNICATION RESEARCH LAB. CU.
ARRAY PROCESSORS CPU
DSP-002

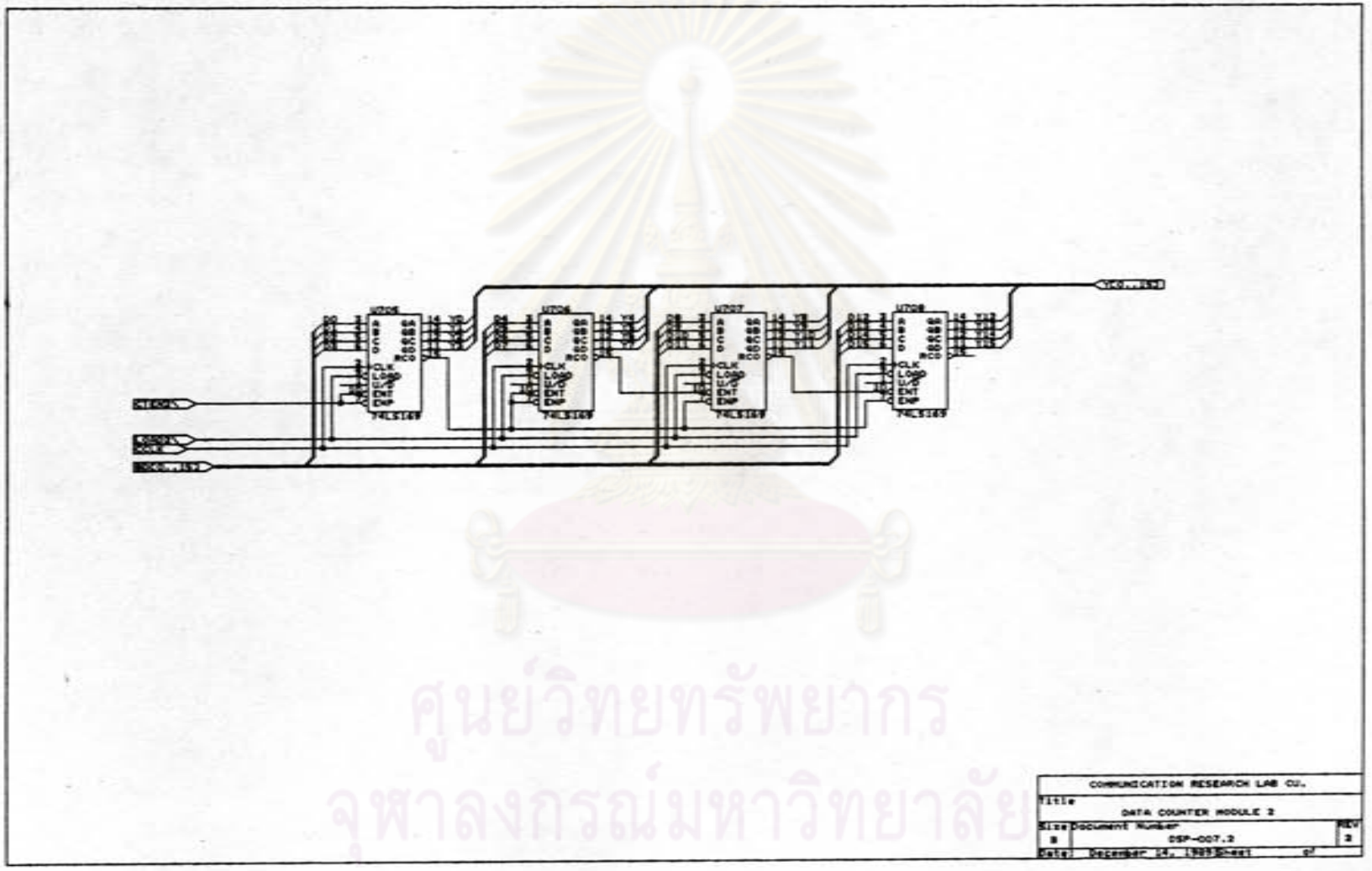

8000-9FFF
COMMUNICATION RESEARCH LAB. OU.
Title
ZSO-COMM.CONTROLLER SHARED MEMORY
Size
B
Document Number
DSP-003
REV
3
Date: December 9, 1993
Sheet 2 of 4

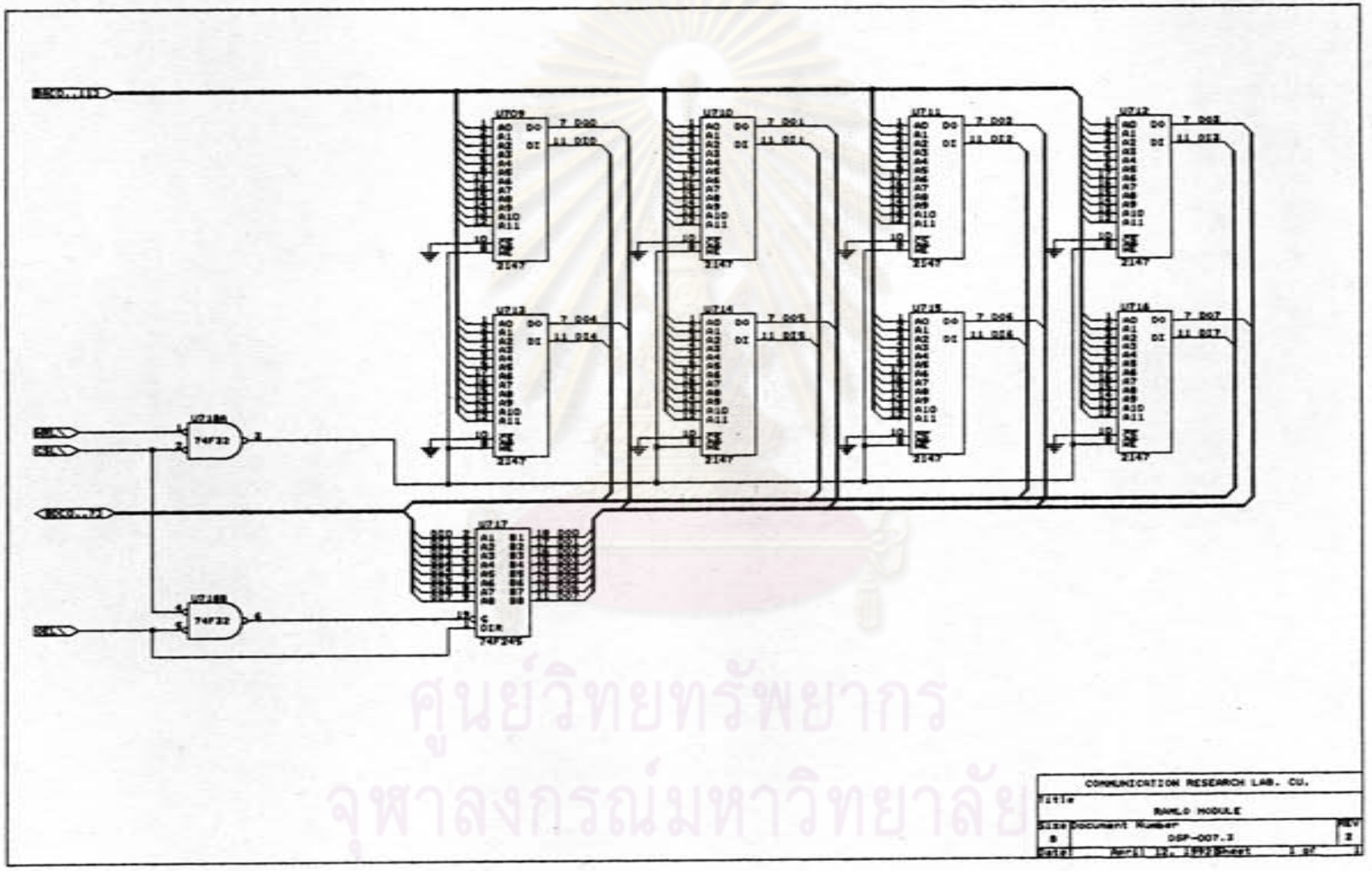

Z80-THS 32010 SHARED MEMORY
ARRAY PROCESSORS CPU
Z80-COMM.CONTROLLER SHARED MEMORY
COMMUNICATION RESEARCH LAB. CU.
Z-80 ARRAY PROCESSOR
DSP-004

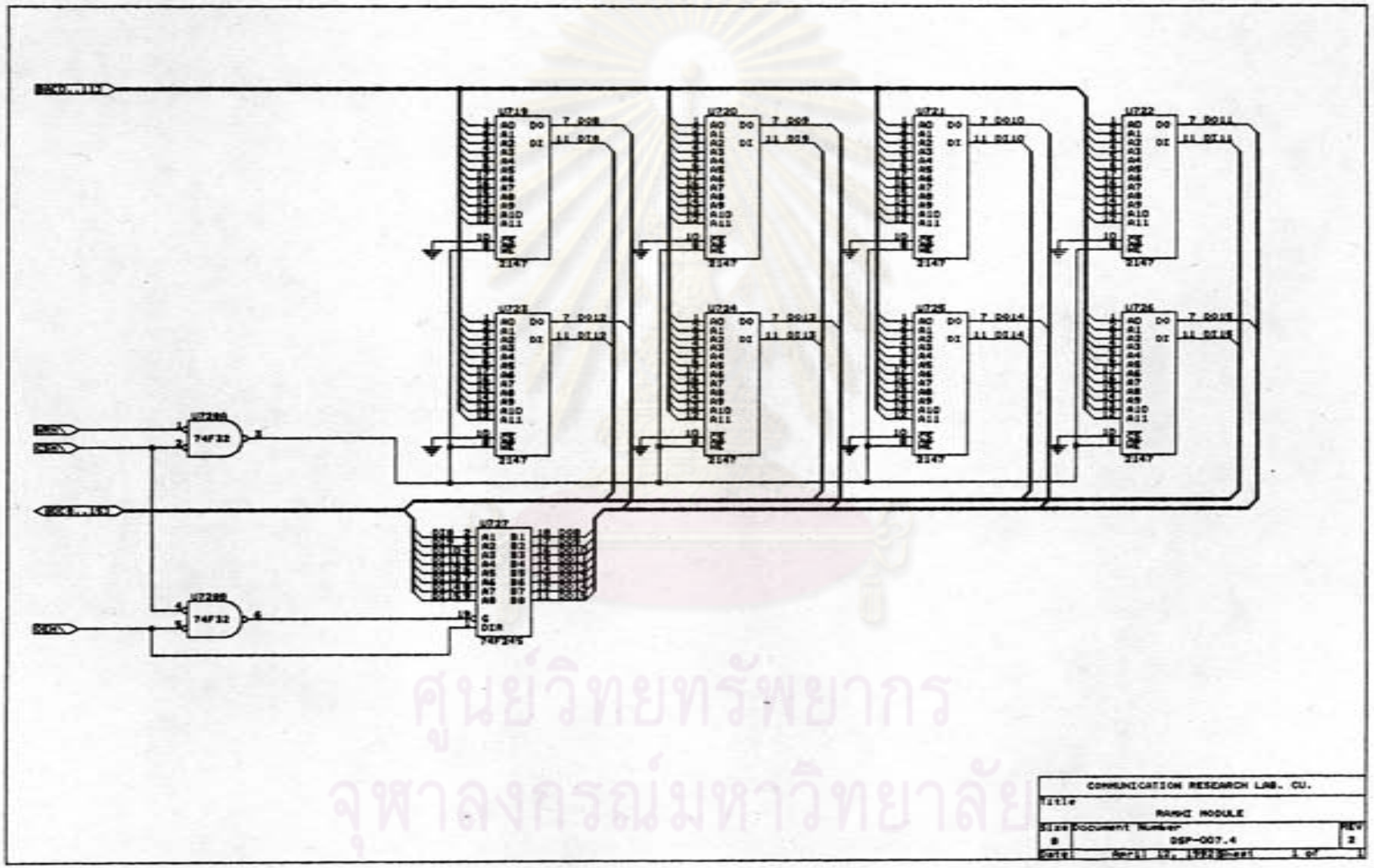


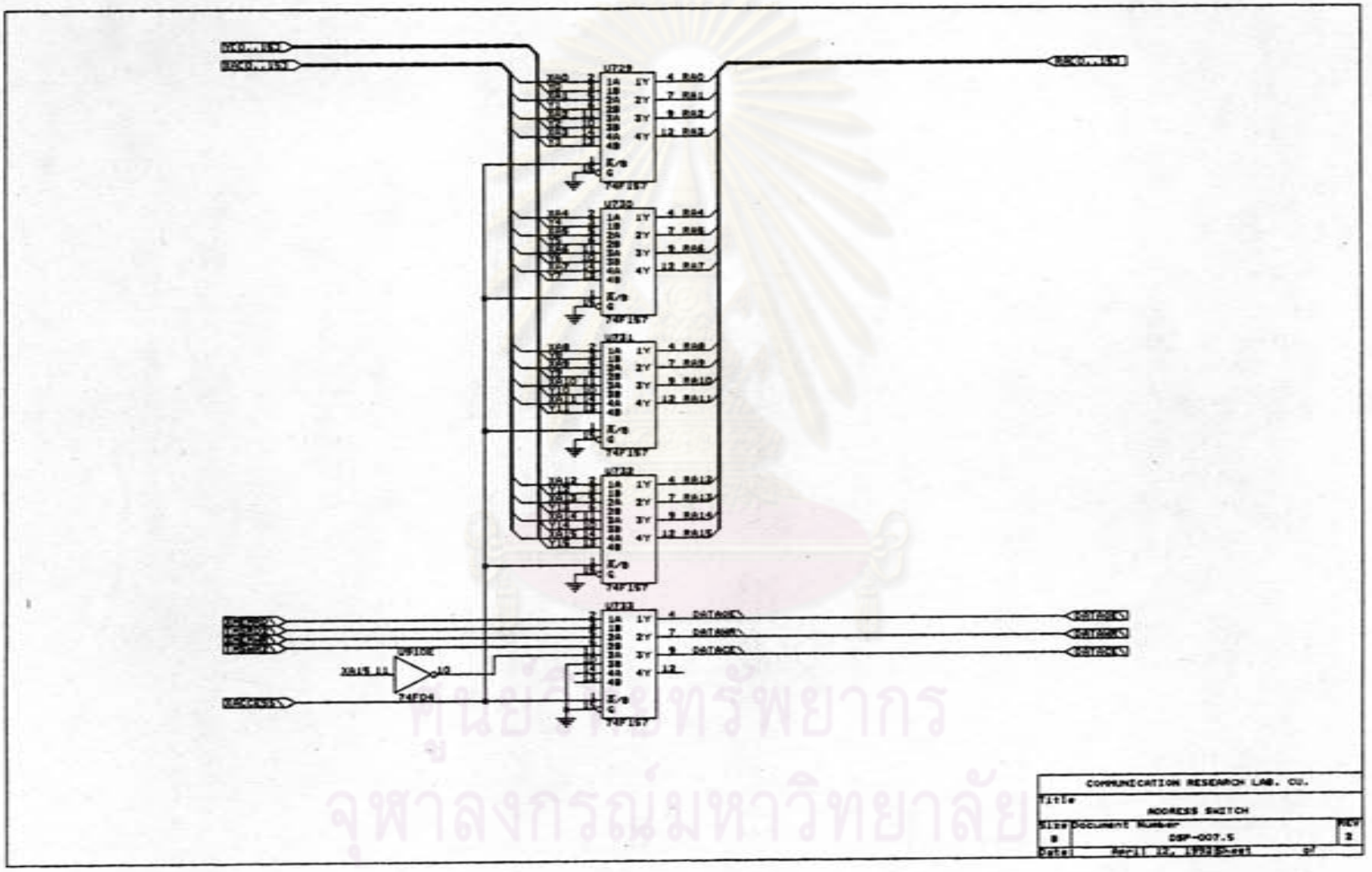


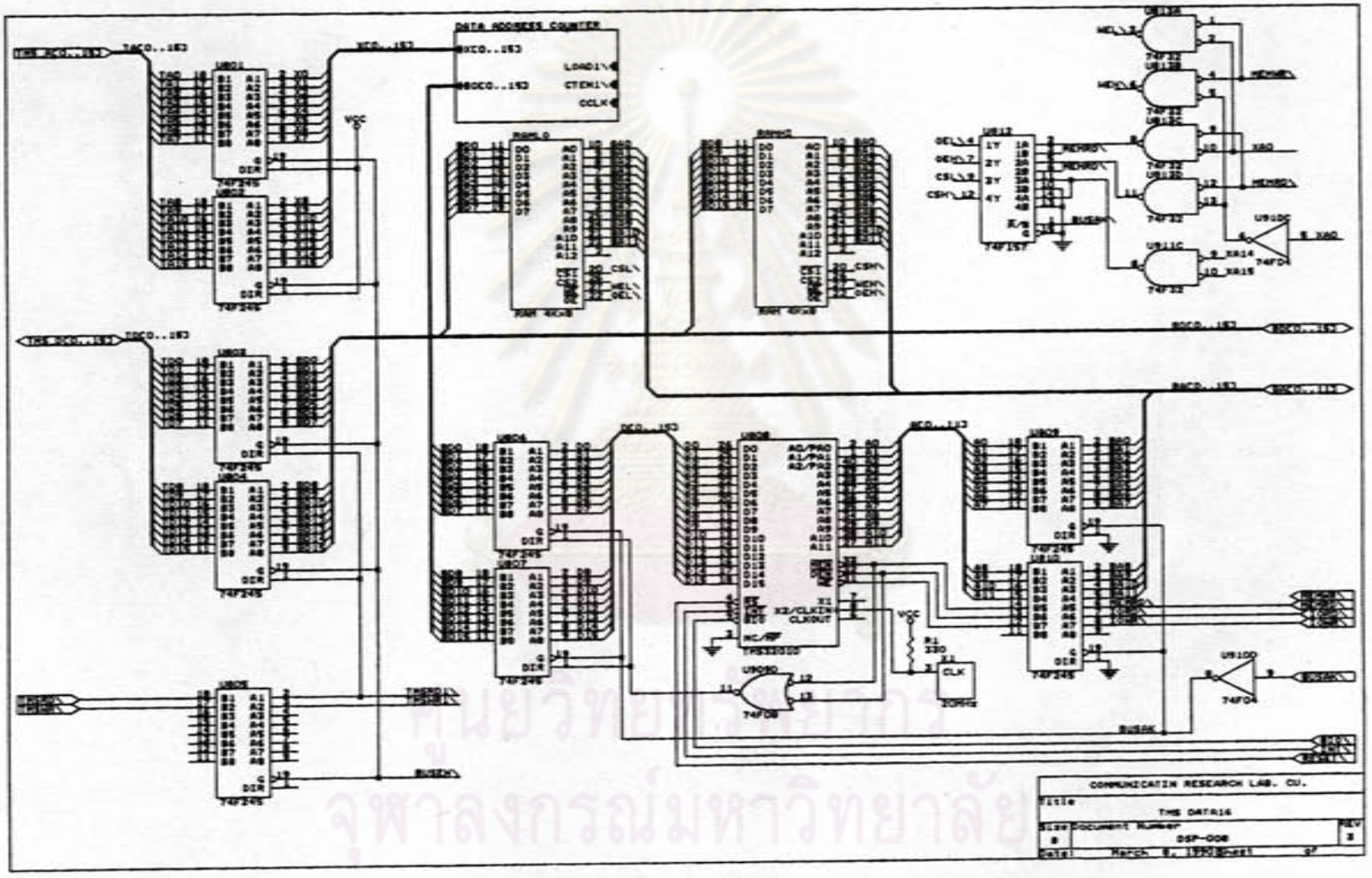


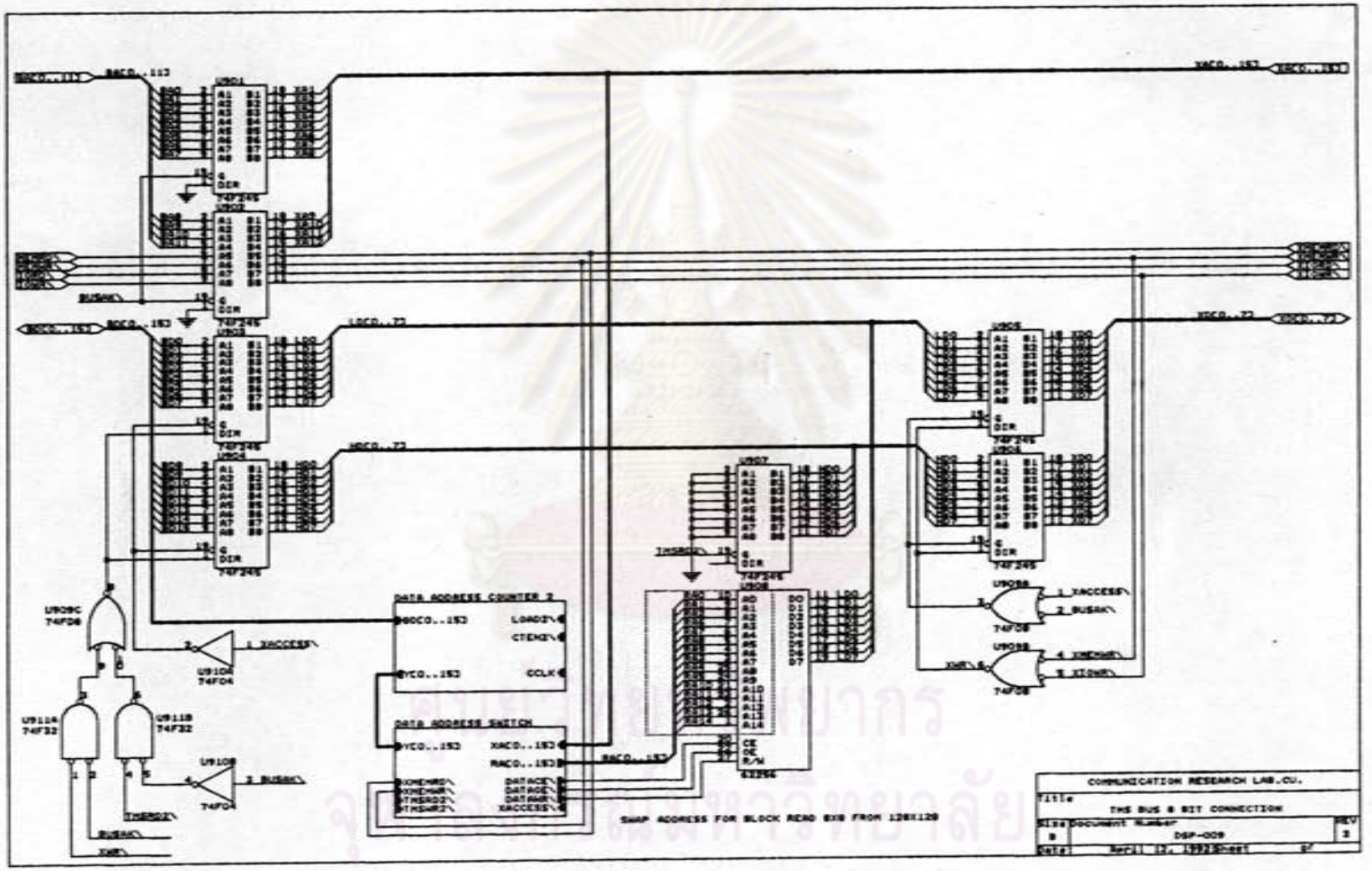


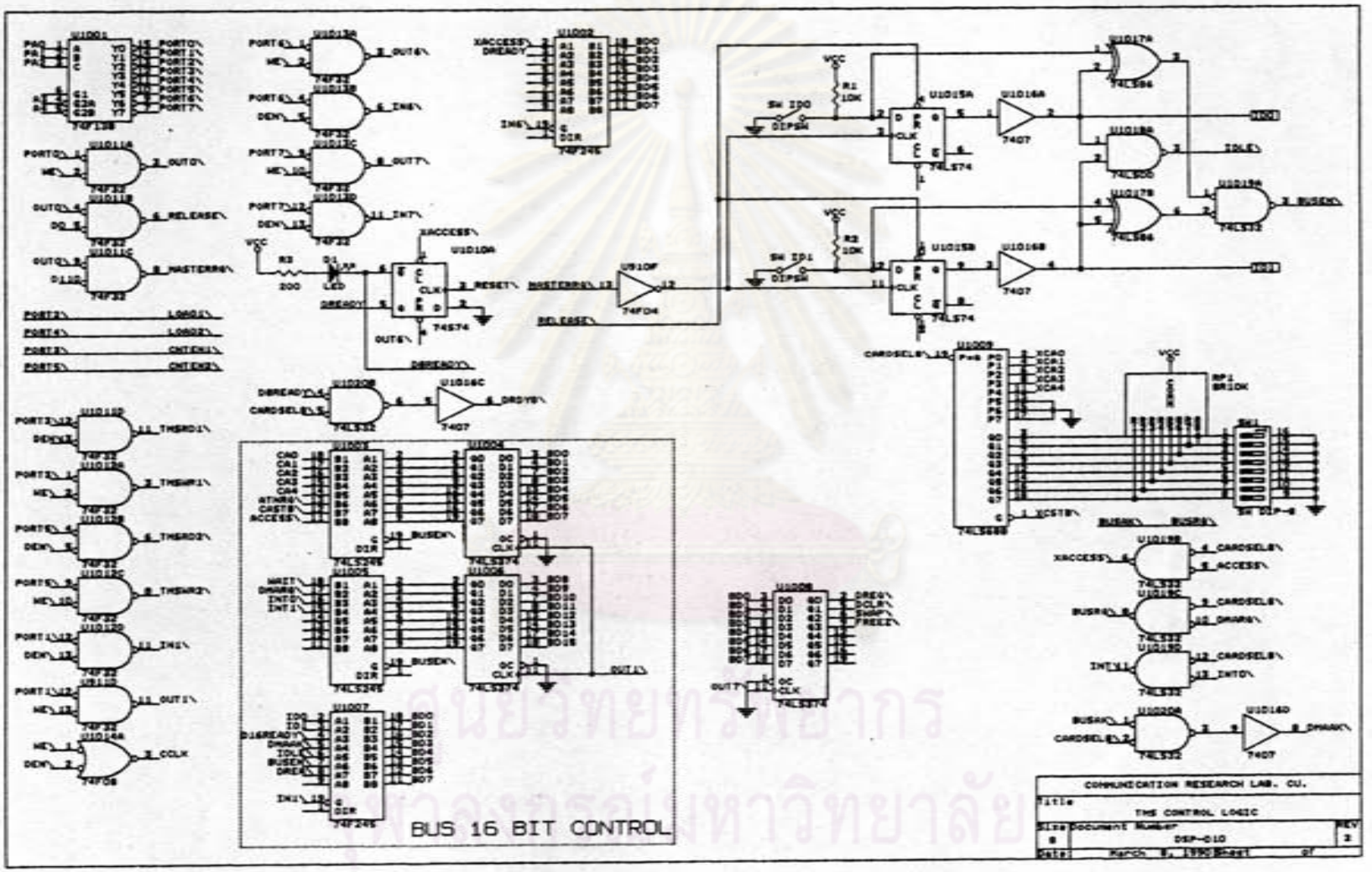









DATA ADDRESS COUNTER 2
DATA ADDRESS SWITCH
SWAP ADDRESS FOR BLOCK READ 8X8 FROM 128X128
COMMUNICATION RESEARCH LAB.,CU.
THE BUS 8 BIT CONNECTION
DSP-009



ภาคผนวก ค

โปรแกรม DCT 8x8 ซึ่งเขียนด้วยภาษาแอสเซมบลีของ TMS32010

```
                IDT     'DCT8X8'
*****************************************************
*----- PROGRAM DCT 8x8 CHEN & SMITH ALGORITHM -----*
*****************************************************
CPI4            EQU     2896    Table for cos & sin
CPI8            EQU     3784    in Q12
SPI8            EQU     1567
CPI16           EQU     4017
SPI16           EQU      799
C3PI16          EQU     3406
S3PI16          EQU     2276

INPORT          EQU     PA0
OUTP            EQU     PA1
*---------- DATA AREA FOR INPUT & OUTPUT----------*
DMA0            EQU     0
DMA1            EQU     1
DMA2            EQU     2
DMA3            EQU     3
DMA4            EQU     4
DMA5            EQU     5
DMA6            EQU     6
DMA7            EQU     7
DMA8            EQU     8
DMA9            EQU     9
DMA10           EQU     10
DMA11           EQU     11
DMA12           EQU     12
DMA13           EQU     13
DMA14           EQU     14
DMA15           EQU     15
DMA16           EQU     16
DMA17           EQU     17
DMA18           EQU     18
DMA19           EQU     19
DMA20           EQU     20
DMA21           EQU     21
DMA22           EQU     22
```

```
DMA23           EQU     23
DMA24           EQU     24
DMA25           EQU     25
DMA26           EQU     26
DMA27           EQU     27
DMA28           EQU     28
DMA29           EQU     29
DMA30           EQU     30
DMA31           EQU     31
DMA32           EQU     32
DMA33           EQU     33
DMA34           EQU     34
DMA35           EQU     35
DMA36           EQU     36
DMA37           EQU     37
DMA38           EQU     38
DMA39           EQU     39
DMA40           EQU     40
DMA41           EQU     41
DMA42           EQU     42
DMA43           EQU     43
DMA44           EQU     44
DMA45           EQU     45
DMA46           EQU     46
DMA47           EQU     47
DMA48           EQU     48
DMA49           EQU     49
DMA50           EQU     50
DMA51           EQU     51
DMA52           EQU     52
DMA53           EQU     53
DMA54           EQU     54
DMA55           EQU     55
DMA56           EQU     56
DMA57           EQU     57
DMA58           EQU     58
DMA59           EQU     59
DMA60           EQU     60
```

```
DMA61           EQU     61
DMA62           EQU     62
DMA63           EQU     63
*-----------------TEMP AREA -----------------*
TMP0            EQU     64
TMP1            EQU     65
TMP2            EQU     66
TMP3            EQU     67
TMP4            EQU     68
TMP5            EQU     69
TMP6            EQU     70
TMP7            EQU     71

DCT             $MACRO  D0,D1,D2,D3,D4,D5,D6,D7
*************************************************
*                  STATE 1                      *
*************************************************
* 0+7 STORE AT TMP
                LAC     :D0:
                ADD     :D7:
                SACL    TMP0
* 1+6
                LAC     :D1:
                ADD     :D6:
                SACL    TMP1
* 2+5
                LAC     :D2:
                ADD     :D5:
                SACL    TMP2
* 3+4
                LAC     :D3:
                ADD     :D4:
                SACL    TMP3
* 3-4
                LAC     :D3:
                SUB     :D4:
                SACL    :D4:
* 2-5
```

```
                LAC     :D2:
                SUB     :D5:
                SACL    TMP5
* 1-6
                LAC     :D1:
                SUB     :D6:
                SACL    TMP6
* 0-7
                LAC     :D0:
                SUB     :D7:
                SACL    :D7:
*************************************************
*                   STATE 2                     *
*************************************************
* 0+3 STORE AT D0
                LAC     TMP0
                ADD     TMP3
                SACL    :D0:
* 1+2
                LAC     TMP1
                ADD     TMP2
                SACL    :D1:
* 1-2
                LAC     TMP1
                SUB     TMP2
                SACL    :D2:
* 0-3
                LAC     TMP0
                SUB     TMP3
                SACL    :D3:

* 6xCOS PI/4 + 5xCOS PI/4  -> 6
                LT      TMP5        LOAD 5
                MPYK    CPI4
                PAC                 ACC = 5xCOS PI/4
                LT      TMP6        LOAD 6
                MPYK    CPI4        P = 6xCOS PI/4
                APAC                ACC = ACC+P
```

```
                SACH    :D6:,4      D6 = ACC >> 12
                PAC                 ACC = P = 6xCOS PI/4
* 6xCOS PI/4 - 5xCOS PI/4  -> 5
                LT      TMP5
                MPYK    CPI4        P = 5xCOS PI/4
                SPAC                ACC = ACC - P
                SACH    :D5:,4      D5 = ACC >> 12

*************************************************
*                  STATE 3                      *
*************************************************
* 4+5
                LAC     :D4:
                ADD     :D5:
                SACL    TMP4
* 4-5
                LAC     :D4:
                SUB     :D5:
                SACL    TMP5
* 7-6
                LAC     :D7:
                SUB     :D6:
                SACL    TMP6
* 6+7
                LAC     :D6:
                ADD     :D7:
                SACL    TMP7
* 0xCOS PI/4 + 1xCOS PI/4
                LT      :D1:        LOAD 1
                MPYK    CPI4
                PAC                 1 -> ACC
                LT      :D0:
                MPYK    CPI4
                APAC                1 + 0
                SACH    :D0:,4
                PAC                 0 -> ACC
* 0xCOS PI/4 - 1xCOS PI/4
                LT      :D1:
```

```
                MPYK    CPI4
                SPAC                    0 - 1
                SACH    :D4:,4
* 3xSIN PI/8 - 2xCOS PI/8
                LT      :D3:
                MPYK    SPI8
                PAC
                LT      :D2:            T = 2
                MPYK    CPI8
                SPAC
                SACH    :D6:,4
* 2xSIN PI/8 + 3xCOS PI/8
                MPYK    SPI8
                PAC
                LT      :D3:
                MPYK    CPI8
                APAC
                SACH    :D2:,4
*****************************************************
*                   STATE 4                         *
*****************************************************
* 4xSIN PI/16 + 7xCOS PI/16   -> 1
                LT      TMP4
                MPYK    SPI16
                PAC
                LT      TMP7
                MPYK    CPI16
                APAC
                SACH    :D1:,4
* 7xSIN PI/16 - 4xCOS PI/16   -> 7
                MPYK    SPI16
                PAC
                LT      TMP4
                MPYK    CPI16
                SPAC
                SACH    :D7:,4
* 5xCOS 3PI/16 + 6xSIN 3PI/16 -> 5
                LT      TMP5
```

```
                MPYK    C3PI16
                PAC
                LT      TMP6
                MPYK    S3PI16
                APAC
                SACH    :D5:,4
* 6xCOS 3PI/16 - 5xSIN 3PI/16 -> 3
                MPYK    C3PI16
                PAC
                LT      TMP5
                MPYK    S3PI16
                SPAC
                SACH    :D3:,4
                $END    DCT

*--------------DIV4 MACRO DIVIDE DATA WITH 4----------*
DIV4            $MACRO  A
                LAC     :A:,14
                SACH    :A:
                $END    DIV4

*----------------- MAIN PROGRAM START ----------------*
                AORG
                B       INIT
                B       INTR
INTR            NOP
INIT            DINT
                ROVM
START           LDPK    0
*----------------- INPUT ROW0 -----------------------*
DATAIN          IN      DMA0,INPORT
                IN      DMA1,INPORT
                IN      DMA2,INPORT
                IN      DMA3,INPORT
                IN      DMA4,INPORT
                IN      DMA5,INPORT
                IN      DMA6,INPORT
                IN      DMA7,INPORT
```

```
*---------------- INPUT ROW1 ----------------*
            IN      DMA8,INPORT
            IN      DMA9,INPORT
            IN      DMA10,INPORT
            IN      DMA11,INPORT
            IN      DMA12,INPORT
            IN      DMA13,INPORT
            IN      DMA14,INPORT
            IN      DMA15,INPORT
*---------------- INPUT ROW2 ----------------*
            IN      DMA16,INPORT
            IN      DMA17,INPORT
            IN      DMA18,INPORT
            IN      DMA19,INPORT
            IN      DMA20,INPORT
            IN      DMA21,INPORT
            IN      DMA22,INPORT
            IN      DMA23,INPORT
*---------------- INPUT ROW3 ----------------*
            IN      DMA24,INPORT
            IN      DMA25,INPORT
            IN      DMA26,INPORT
            IN      DMA27,INPORT
            IN      DMA28,INPORT
            IN      DMA29,INPORT
            IN      DMA30,INPORT
            IN      DMA31,INPORT
*---------------- INPUT ROW4 ----------------*
            IN      DMA32,INPORT
            IN      DMA33,INPORT
            IN      DMA34,INPORT
            IN      DMA35,INPORT
            IN      DMA36,INPORT
            IN      DMA37,INPORT
            IN      DMA38,INPORT
            IN      DMA39,INPORT
*---------------- INPUT ROW5 ----------------*
            IN      DMA40,INPORT
```

```
                IN      DMA41,INPORT
                IN      DMA42,INPORT
                IN      DMA43,INPORT
                IN      DMA44,INPORT
                IN      DMA45,INPORT
                IN      DMA46,INPORT
                IN      DMA47,INPORT
*---------------- INPUT ROW6 ----------------*
                IN      DMA48,INPORT
                IN      DMA49,INPORT
                IN      DMA50,INPORT
                IN      DMA51,INPORT
                IN      DMA52,INPORT
                IN      DMA53,INPORT
                IN      DMA54,INPORT
                IN      DMA55,INPORT
*---------------- INPUT ROW7 ----------------*
                IN      DMA56,INPORT
                IN      DMA57,INPORT
                IN      DMA58,INPORT
                IN      DMA59,INPORT
                IN      DMA60,INPORT
                IN      DMA61,INPORT
                IN      DMA62,INPORT
                IN      DMA63,INPORT
*
    DCT  DMA0,DMA1,DMA2,DMA3,DMA4,DMA5,DMA6,DMA7
    DCT  DMA8,DMA9,DMA10,DMA11,DMA12,DMA13,DMA14,DMA15
    DCT  DMA16,DMA17,DMA18,DMA19,DMA20,DMA21,DMA22,DMA23
    DCT  DMA24,DMA25,DMA26,DMA27,DMA28,DMA29,DMA30,DMA31
    DCT  DMA32,DMA33,DMA34,DMA35,DMA36,DMA37,DMA38,DMA39
    DCT  DMA40,DMA41,DMA42,DMA43,DMA44,DMA45,DMA46,DMA47
    DCT  DMA48,DMA49,DMA50,DMA51,DMA52,DMA53,DMA54,DMA55
    DCT  DMA56,DMA57,DMA58,DMA59,DMA60,DMA61,DMA62,DMA63
    DCT  DMA0,DMA8,DMA16,DMA24,DMA32,DMA40,DMA48,DMA56
    DCT  DMA1,DMA9,DMA17,DMA25,DMA33,DMA41,DMA49,DMA57
    DCT  DMA2,DMA10,DMA18,DMA26,DMA34,DMA42,DMA50,DMA58
    DCT  DMA3,DMA11,DMA19,DMA27,DMA35,DMA43,DMA51,DMA59
```

```
     DCT  DMA4,DMA12,DMA20,DMA28,DMA36,DMA44,DMA52,DMA60
     DCT  DMA5,DMA13,DMA21,DMA29,DMA37,DMA45,DMA53,DMA61
     DCT  DMA6,DMA14,DMA22,DMA30,DMA38,DMA46,DMA54,DMA62
     DCT  DMA7,DMA15,DMA23,DMA31,DMA39,DMA47,DMA55,DMA63
*------------------ SCALE DATA WITH 4------------------*
SCALE          DIV4    DMA0
               DIV4    DMA1
               DIV4    DMA2
               DIV4    DMA3
               DIV4    DMA4
               DIV4    DMA5
               DIV4    DMA6
               DIV4    DMA7
               DIV4    DMA8
               DIV4    DMA9
               DIV4    DMA10
               DIV4    DMA11
               DIV4    DMA12
               DIV4    DMA13
               DIV4    DMA14
               DIV4    DMA15
               DIV4    DMA16
               DIV4    DMA17
               DIV4    DMA18
               DIV4    DMA19
               DIV4    DMA20
               DIV4    DMA21
               DIV4    DMA22
               DIV4    DMA23
               DIV4    DMA24
               DIV4    DMA25
               DIV4    DMA26
               DIV4    DMA27
               DIV4    DMA28
               DIV4    DMA29
               DIV4    DMA30
               DIV4    DMA31
               DIV4    DMA32
```

```
                DIV4    DMA33
                DIV4    DMA34
                DIV4    DMA35
                DIV4    DMA36
                DIV4    DMA37
                DIV4    DMA38
                DIV4    DMA39
                DIV4    DMA40
                DIV4    DMA41
                DIV4    DMA42
                DIV4    DMA43
                DIV4    DMA44
                DIV4    DMA45
                DIV4    DMA46
                DIV4    DMA47
                DIV4    DMA48
                DIV4    DMA49
                DIV4    DMA50
                DIV4    DMA51
                DIV4    DMA52
                DIV4    DMA53
                DIV4    DMA54
                DIV4    DMA55
                DIV4    DMA56
                DIV4    DMA57
                DIV4    DMA58
                DIV4    DMA59
                DIV4    DMA60
                DIV4    DMA61
                DIV4    DMA62
                DIV4    DMA63
*---------- OUTPUT DATA WITH ZIGZAG SCAN ----------*
*------------------ OUTPUT ROW0 ---------------------*
ZIGOUT          OUT     DMA0,OUTP
                OUT     DMA1,OUTP
                OUT     DMA8,OUTP
                OUT     DMA16,OUTP
                OUT     DMA9,OUTP
```

```
                OUT     DMA2,OUTP
                OUT     DMA3,OUTP
                OUT     DMA10,OUTP
*---------------- OUTPUT ROW1 ----------------*
                OUT     DMA17,OUTP
                OUT     DMA24,OUTP
                OUT     DMA32,OUTP
                OUT     DMA25,OUTP
                OUT     DMA18,OUTP
                OUT     DMA11,OUTP
                OUT     DMA4,OUTP
                OUT     DMA5,OUTP
*---------------- OUTPUT ROW2 ----------------*
                OUT     DMA12,OUTP
                OUT     DMA19,OUTP
                OUT     DMA26,OUTP
                OUT     DMA33,OUTP
                OUT     DMA40,OUTP
                OUT     DMA48,OUTP
                OUT     DMA41,OUTP
                OUT     DMA34,OUTP
*---------------- OUTPUT ROW3 ----------------*
                OUT     DMA27,OUTP
                OUT     DMA20,OUTP
                OUT     DMA13,OUTP
                OUT     DMA6,OUTP
                OUT     DMA7,OUTP
                OUT     DMA14,OUTP
                OUT     DMA21,OUTP
                OUT     DMA28,OUTP
*---------------- OUTPUT ROW4 ----------------*
                OUT     DMA35,OUTP
                OUT     DMA42,OUTP
                OUT     DMA49,OUTP
                OUT     DMA56,OUTP
                OUT     DMA57,OUTP
                OUT     DMA50,OUTP
                OUT     DMA43,OUTP
```

```
                OUT     DMA36,OUTP
*---------------- OUTPUT ROW5 ----------------*
                OUT     DMA29,OUTP
                OUT     DMA22,OUTP
                OUT     DMA15,OUTP
                OUT     DMA23,OUTP
                OUT     DMA30,OUTP
                OUT     DMA37,OUTP
                OUT     DMA44,OUTP
                OUT     DMA51,OUTP
*---------------- OUTPUT ROW6 ----------------*
                OUT     DMA58,OUTP
                OUT     DMA59,OUTP
                OUT     DMA52,OUTP
                OUT     DMA45,OUTP
                OUT     DMA38,OUTP
                OUT     DMA31,OUTP
                OUT     DMA39,OUTP
                OUT     DMA46,OUTP
*---------------- OUTPUT ROW7 ----------------*
                OUT     DMA53,OUTP
                OUT     DMA60,OUTP
                OUT     DMA61,OUTP
                OUT     DMA54,OUTP
                OUT     DMA47,OUTP
                OUT     DMA55,OUTP
                OUT     DMA62,OUTP
                OUT     DMA63,OUTP
                B       START
                END
```

ภาคผนวก ง

โปรแกรม DCT^{-1} 8x8 ซึ่งเขียนด้วยภาษาแอสเซมบลีของ TMS32010

```
                IDT     'IDCT8X8'
CPI4            EQU     2896    Table for cos & sin
CPI8            EQU     3784    in Q12
SPI8            EQU     1567
CPI16           EQU     4017
SPI16           EQU      799
C3PI16          EQU     3406
S3PI16          EQU     2276

INPORT          EQU     PA0
OUTP            EQU     PA1
*---------- DATA AREA FOR INPUT & OUTPUT----------*
DMA0            EQU     0
DMA1            EQU     1
DMA2            EQU     2
DMA3            EQU     3
DMA4            EQU     4
DMA5            EQU     5
DMA6            EQU     6
DMA7            EQU     7
DMA8            EQU     8
DMA9            EQU     9
DMA10           EQU     10
DMA11           EQU     11
DMA12           EQU     12
DMA13           EQU     13
DMA14           EQU     14
DMA15           EQU     15
DMA16           EQU     16
DMA17           EQU     17
DMA18           EQU     18
DMA19           EQU     19
DMA20           EQU     20
DMA21           EQU     21
DMA22           EQU     22
DMA23           EQU     23
DMA24           EQU     24
DMA25           EQU     25
```

```
DMA26           EQU     26
DMA27           EQU     27
DMA28           EQU     28
DMA29           EQU     29
DMA30           EQU     30
DMA31           EQU     31
DMA32           EQU     32
DMA33           EQU     33
DMA34           EQU     34
DMA35           EQU     35
DMA36           EQU     36
DMA37           EQU     37
DMA38           EQU     38
DMA39           EQU     39
DMA40           EQU     40
DMA41           EQU     41
DMA42           EQU     42
DMA43           EQU     43
DMA44           EQU     44
DMA45           EQU     45
DMA46           EQU     46
DMA47           EQU     47
DMA48           EQU     48
DMA49           EQU     49
DMA50           EQU     50
DMA51           EQU     51
DMA52           EQU     52
DMA53           EQU     53
DMA54           EQU     54
DMA55           EQU     55
DMA56           EQU     56
DMA57           EQU     57
DMA58           EQU     58
DMA59           EQU     59
DMA60           EQU     60
DMA61           EQU     61
DMA62           EQU     62
DMA63           EQU     63
```

```
*--------------------TEMP AREA ---------------------*
TMP0            EQU     64
TMP1            EQU     65
TMP2            EQU     66
TMP3            EQU     67
TMP4            EQU     68
TMP5            EQU     69
TMP6            EQU     70
TMP7            EQU     71

IDCT            $MACRO   D0,D1,D2,D3,D4,D5,D6,D7
*************************************************
*                   STATE 1                     *
*************************************************
* 4xSIN PI/16 - 7xCOS PI/16   -> 4
                LT      :D1:
                MPYK    SPI16
                PAC
                LT      :D7:
                MPYK    CPI16
                SPAC
                SACH    TMP4,4
* 7xSIN PI/16 + 4xCOS PI/16   -> 7
                MPYK    SPI16
                PAC
                LT      :D1:
                MPYK    CPI16
                APAC
                SACH    TMP7,4
* 5xCOS 3PI/16 - 6xSIN 3PI/16 -> 5
                LT      :D5:
                MPYK    C3PI16
                PAC
                LT      :D3:
                MPYK    S3PI16
                SPAC
                SACH    TMP5,4
* 6xCOS 3PI/16 + 5xSIN 3PI/16 -> 6
```

```
                MPYK    C3PI16
                PAC
                LT      :D5:
                MPYK    S3PI16
                APAC
                SACH    TMP6,4
*****************************************************
*                   STATE 2                         *
*****************************************************
* 0xCOS PI/4 + 1xCOS PI/4
                LT      :D4:            LOAD 1
                MPYK    CPI4
                PAC                     1xCOS PI/4 -> ACC
                LT      :D0:
                MPYK    CPI4
                APAC                    1 + 0
                SACH    :D0:,4
                PAC                     0xCOS PI/4 -> ACC
* 0xCOS PI/4 - 1xCOS PI/4
                LT      :D4:
                MPYK    CPI4
                SPAC                    0 - 1
                SACH    :D1:,4
* 3xSIN PI/8 + 2xCOS PI/8 -> 3
                LT      :D6:
                MPYK    SPI8
                PAC
                LT      :D2:
                MPYK    CPI8
                APAC
                SACH    :D3:,4
* 2xSIN PI/8 - 3xCOS PI/8  -> 2
                MPYK    SPI8
                PAC
                LT      :D6:            T = 2
                MPYK    CPI8
                SPAC
                SACH    :D2:,4
```

```
* 4+5
                LAC     TMP4
                ADD     TMP5
                SACL    :D4:
* 4-5
                LAC     TMP4
                SUB     TMP5
                SACL    :D5:
* 7-6
                LAC     TMP7
                SUB     TMP6
                SACL    :D6:
* 6+7
                LAC     TMP6
                ADD     TMP7
                SACL    :D7:
*************************************************
*                   STATE 3                     *
*************************************************
* 0+3 STORE AT 0
                LAC     :D0:
                ADD     :D3:
                SACL    TMP0
* 1+2
                LAC     :D1:
                ADD     :D2:
                SACL    TMP1
* 1-2
                LAC     :D1:
                SUB     :D2:
                SACL    TMP2
* 0-3
                LAC     :D0:
                SUB     :D3:
                SACL    TMP3

* 6xCOS PI/4 + 5xCOS PI/4  -> 6
                LT      :D5:        LOAD 5
```

```
                MPYK    CPI4
                PAC                 ACC = 5xCOS PI/4
                LT      :D6:          LOAD 6
                MPYK    CPI4        P = 6xCOS PI/4
                APAC                6 + 5
                SACH    TMP6,4
                PAC                 6xCOS PI/4 -> ACC
* 6xCOS PI/4 - 5xCOS PI/4  -> 5
                LT      :D5:
                MPYK    CPI4
                SPAC
                SACH    TMP5,4

**************************************************
*                  STATE 4                       *
**************************************************
* 0+7 STORE AT 0
                LAC     TMP0
                ADD     :D7:
                SACL    :D0:
* 1+6
                LAC     TMP1
                ADD     TMP6
                SACL    :D1:
* 2+5
                LAC     TMP2
                ADD     TMP5
                SACL    :D2:
* 3+4
                LAC     TMP3
                ADD     TMP4
                SACL    :D3:
* 3-4
                LAC     TMP3
                SUB     TMP4
                SACL    :D4:
* 2-5
                LAC     TMP2
```

```
                SUB     TMP5
                SACL    :D5:
* 1-6
                LAC     TMP1
                SUB     TMP6
                SACL    :D6:
* 0-7
                LAC     TMP0
                SUB     :D7:
                SACL    :D7:
                $END

DIV4            $MACRO  A
                LAC     :A:,14
                SACH    :A:
                $END

                AORG
                B       INIT
                B       INTR
INTR            NOP
INIT            DINT
                ROVM
START           LDPK    0
*---------- INPUT DATA WITH ZIGZAG SCAN ----------*
*---------------- INPUT ROW0 ----------------*
ZIGIN           IN      DMA0,INPORT
                IN      DMA1,INPORT
                IN      DMA8,INPORT
                IN      DMA16,INPORT
                IN      DMA9,INPORT
                IN      DMA2,INPORT
                IN      DMA3,INPORT
                IN      DMA10,INPORT
*---------------- INPUT ROW1 ----------------*
                IN      DMA17,INPORT
                IN      DMA24,INPORT
                IN      DMA32,INPORT
```

```
                IN      DMA25,INPORT
                IN      DMA18,INPORT
                IN      DMA11,INPORT
                IN      DMA4,INPORT
                IN      DMA5,INPORT
*---------------- INPUT ROW2 ----------------*
                IN      DMA12,INPORT
                IN      DMA19,INPORT
                IN      DMA26,INPORT
                IN      DMA33,INPORT
                IN      DMA40,INPORT
                IN      DMA48,INPORT
                IN      DMA41,INPORT
                IN      DMA34,INPORT
*---------------- INPUT ROW3 ----------------*
                IN      DMA27,INPORT
                IN      DMA20,INPORT
                IN      DMA13,INPORT
                IN      DMA6,INPORT
                IN      DMA7,INPORT
                IN      DMA14,INPORT
                IN      DMA21,INPORT
                IN      DMA28,INPORT
*---------------- INPUT ROW4 ----------------*
                IN      DMA35,INPORT
                IN      DMA42,INPORT
                IN      DMA49,INPORT
                IN      DMA56,INPORT
                IN      DMA57,INPORT
                IN      DMA50,INPORT
                IN      DMA43,INPORT
                IN      DMA36,INPORT
*---------------- INPUT ROW5 ----------------*
                IN      DMA29,INPORT
                IN      DMA22,INPORT
                IN      DMA15,INPORT
                IN      DMA23,INPORT
                IN      DMA30,INPORT
```

```
                IN      DMA37,INPORT
                IN      DMA44,INPORT
                IN      DMA51,INPORT
*---------------- INPUT ROW6 ----------------*
                IN      DMA58,INPORT
                IN      DMA59,INPORT
                IN      DMA52,INPORT
                IN      DMA45,INPORT
                IN      DMA38,INPORT
                IN      DMA31,INPORT
                IN      DMA39,INPORT
                IN      DMA46,INPORT
*---------------- INPUT ROW7 ----------------*
                IN      DMA53,INPORT
                IN      DMA60,INPORT
                IN      DMA61,INPORT
                IN      DMA54,INPORT
                IN      DMA47,INPORT
                IN      DMA55,INPORT
                IN      DMA62,INPORT
                IN      DMA63,INPORT
                B       TRANS
*
TRANS IDCT  DMA0,DMA1,DMA2,DMA3,DMA4,DMA5,DMA6,DMA7
      IDCT  DMA8,DMA9,DMA10,DMA11,DMA12,DMA13,DMA14,DMA15
      IDCT  DMA16,DMA17,DMA18,DMA19,DMA20,DMA21,DMA22,DMA23
      IDCT  DMA24,DMA25,DMA26,DMA27,DMA28,DMA29,DMA30,DMA31
      IDCT  DMA32,DMA33,DMA34,DMA35,DMA36,DMA37,DMA38,DMA39
      IDCT  DMA40,DMA41,DMA42,DMA43,DMA44,DMA45,DMA46,DMA47
      IDCT  DMA48,DMA49,DMA50,DMA51,DMA52,DMA53,DMA54,DMA55
      IDCT  DMA56,DMA57,DMA58,DMA59,DMA60,DMA61,DMA62,DMA63
      IDCT  DMA0,DMA8,DMA16,DMA24,DMA32,DMA40,DMA48,DMA56
      IDCT  DMA1,DMA9,DMA17,DMA25,DMA33,DMA41,DMA49,DMA57
      IDCT  DMA2,DMA10,DMA18,DMA26,DMA34,DMA42,DMA50,DMA58
      IDCT  DMA3,DMA11,DMA19,DMA27,DMA35,DMA43,DMA51,DMA59
      IDCT  DMA4,DMA12,DMA20,DMA28,DMA36,DMA44,DMA52,DMA60
      IDCT  DMA5,DMA13,DMA21,DMA29,DMA37,DMA45,DMA53,DMA61
      IDCT  DMA6,DMA14,DMA22,DMA30,DMA38,DMA46,DMA54,DMA62
```

```
IDCT  DMA7,DMA15,DMA23,DMA31,DMA39,DMA47,DMA55,DMA63
          DIV4    DMA0
          DIV4    DMA1
          DIV4    DMA2
          DIV4    DMA3
          DIV4    DMA4
          DIV4    DMA5
          DIV4    DMA6
          DIV4    DMA7
          DIV4    DMA8
          DIV4    DMA9
          DIV4    DMA10
          DIV4    DMA11
          DIV4    DMA12
          DIV4    DMA13
          DIV4    DMA14
          DIV4    DMA15
          DIV4    DMA16
          DIV4    DMA17
          DIV4    DMA18
          DIV4    DMA19
          DIV4    DMA20
          DIV4    DMA21
          DIV4    DMA22
          DIV4    DMA23
          DIV4    DMA24
          DIV4    DMA25
          DIV4    DMA26
          DIV4    DMA27
          DIV4    DMA28
          DIV4    DMA29
          DIV4    DMA30
          DIV4    DMA31
          DIV4    DMA32
          DIV4    DMA33
          DIV4    DMA34
          DIV4    DMA35
          DIV4    DMA36
```

```
                DIV4    DMA37
                DIV4    DMA38
                DIV4    DMA39
                DIV4    DMA40
                DIV4    DMA41
                DIV4    DMA42
                DIV4    DMA43
                DIV4    DMA44
                DIV4    DMA45
                DIV4    DMA46
                DIV4    DMA47
                DIV4    DMA48
                DIV4    DMA49
                DIV4    DMA50
                DIV4    DMA51
                DIV4    DMA52
                DIV4    DMA53
                DIV4    DMA54
                DIV4    DMA55
                DIV4    DMA56
                DIV4    DMA57
                DIV4    DMA58
                DIV4    DMA59
                DIV4    DMA60
                DIV4    DMA61
                DIV4    DMA62
                DIV4    DMA63
*
*---------------- OUTPUT ROW0 ----------------*
DATAOUT         OUT     DMA0,OUTP
                OUT     DMA1,OUTP
                OUT     DMA2,OUTP
                OUT     DMA3,OUTP
                OUT     DMA4,OUTP
                OUT     DMA5,OUTP
                OUT     DMA6,OUTP
                OUT     DMA7,OUTP
*---------------- OUTPUT ROW1 ----------------*
```

```
                OUT     DMA8,OUTP
                OUT     DMA9,OUTP
                OUT     DMA10,OUTP
                OUT     DMA11,OUTP
                OUT     DMA12,OUTP
                OUT     DMA13,OUTP
                OUT     DMA14,OUTP
                OUT     DMA15,OUTP
*---------------- OUTPUT ROW2 ----------------*
                OUT     DMA16,OUTP
                OUT     DMA17,OUTP
                OUT     DMA18,OUTP
                OUT     DMA19,OUTP
                OUT     DMA20,OUTP
                OUT     DMA21,OUTP
                OUT     DMA22,OUTP
                OUT     DMA23,OUTP
*---------------- OUTPUT ROW3 ----------------*
                OUT     DMA24,OUTP
                OUT     DMA25,OUTP
                OUT     DMA26,OUTP
                OUT     DMA27,OUTP
                OUT     DMA28,OUTP
                OUT     DMA29,OUTP
                OUT     DMA30,OUTP
                OUT     DMA31,OUTP
*---------------- OUTPUT ROW4 ----------------*
                OUT     DMA32,OUTP
                OUT     DMA33,OUTP
                OUT     DMA34,OUTP
                OUT     DMA35,OUTP
                OUT     DMA36,OUTP
                OUT     DMA37,OUTP
                OUT     DMA38,OUTP
                OUT     DMA39,OUTP
*---------------- OUTPUT ROW5 ----------------*
                OUT     DMA40,OUTP
                OUT     DMA41,OUTP
```

```
                OUT     DMA42,OUTP
                OUT     DMA43,OUTP
                OUT     DMA44,OUTP
                OUT     DMA45,OUTP
                OUT     DMA46,OUTP
                OUT     DMA47,OUTP
*---------------- OUTPUT ROW6 ----------------*
                OUT     DMA48,OUTP
                OUT     DMA49,OUTP
                OUT     DMA50,OUTP
                OUT     DMA51,OUTP
                OUT     DMA52,OUTP
                OUT     DMA53,OUTP
                OUT     DMA54,OUTP
                OUT     DMA55,OUTP
*---------------- OUTPUT ROW7 ----------------*
                OUT     DMA56,OUTP
                OUT     DMA57,OUTP
                OUT     DMA58,OUTP
                OUT     DMA59,OUTP
                OUT     DMA60,OUTP
                OUT     DMA61,OUTP
                OUT     DMA62,OUTP
                OUT     DMA63,OUTP
                END
```

ประวัติผู้เขียน

นาย ธีระยุทธ บุญโชติ เกิดเมื่อวันที่ 20 มิถุนายน พ.ศ. 2507 ที่กรุงเทพมหานครฯ สำเร็จการศึกษาระดับปริญญาวิศวกรรมศาสตรบัณฑิตสาขาวิศวกรรมไฟฟ้า จาก คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปี พ.ศ. 2529 และได้เข้าศึกษาต่อในระดับปริญญามหาบัณฑิต สาขาวิศวกรรมไฟฟ้า สังกัดห้องปฏิบัติการวิจัยระบบไฟฟ้าสื่อสาร คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย โดยมีความสนใจทางด้าน Digital Signal Processing, การประมวลผลภาพ และ การสื่อสารข้อมูล ปัจจุบันเป็นพนักงานบริษัท พรีเมียร์เอ็นเตอร์ไพรซ์ จำกัด ตำแหน่ง ผู้จัดการฝ่ายเทคนิค