

Appendix F

Software listing for the three phase cycloconverter

The software is written mostly in the C programming language with some routines written in assembler language for the TMS320C31 digital signal processor. For a guide to the pseudo code documentation, see Appendix G.

The following equation is used as the double integral algorithm in the software. The terms are labelled for identification.

$$\begin{aligned}
 0 = & \int_{t_0}^{t_2} \int_{t_0}^t v_i dt^2 + K(t_2 - t_1) \int_{t_0}^{t_2} v_i dt \\
 & \quad 1(a) \qquad \qquad 1(c) \\
 & - \int_{t_1}^{t_2} \int_{t_1}^t v_b dt^2 - K(t_2 - t_1) \int_{t_1}^{t_2} v_b dt \\
 & \quad 1(e) \qquad \qquad 1(f) \\
 & - \int_{t_1}^{t_2} \psi(t) dt - K(t_2 - t_1) [\psi(t_2) - \psi(t_1)] \\
 & \quad 1(b) \qquad \qquad 1(d) \\
 & - K(t_2 - t_1) \int^{t_1} (v_o - v_b) dt \\
 & \quad 2 \\
 & + \int_{t_1}^{t_f} \int_{t_0}^t (v_o - v_b) dt^2 + (t_2 - t_f) \int_{t_0}^{t_f} (v_o - v_b) dt + K(t_2 - t_1) \int_{t_0}^{t_f} (v_o - v_b) dt \\
 & \quad 3(a) \qquad \qquad 4(a) \qquad \qquad 4(b) \\
 & - \int_{t_0}^{t_f} \int_{t_0}^t v_i dt^2 - (t_2 - t_f) \int_{t_0}^{t_f} v_i dt - K(t_2 - t_1) \int_{t_0}^{t_f} v_i dt \\
 & \quad 3(b) \qquad \qquad 4(c) \qquad \qquad 4(d) \\
 & + \int_{t_1}^{t_f} \int_{t_1}^t v_b dt^2 + (t_2 - t_f) \int_{t_1}^{t_f} v_b dt + K(t_2 - t_1) \int_{t_1}^{t_f} v_b dt \\
 & \quad 3(c) \qquad \qquad 4(e) \qquad \qquad 4(f)
 \end{aligned}$$

```

/* CYCLO3.H */

/* Header file for CYCLO3 program */

/* Turn listing off
asm(" .nolist");*/

/* definitions */
#define SAMP 120 /* samples per mains period */
#define SAMPT 40 /* samples per mains period/3 */
#define FLUXMAX 250.0 /* max value in flux registers */
#define SPEED_POSMAX 0x18000 /* max positive speed. 0x10000 X 180/SAMP
= 30Hz */
#define SPEED_NEGMAX 0x18000 /* max negative speed. 0x10000 X 180/SAMP
= 30Hz */

/* Function Prototypes */
void pll_init(void);
void pll_wait_mains_pos(void);
void pll_pos(void);
void pll_start(void);
void c_int09(void);
void pll_calculate(void);
void read_init(void);
void read_start(void);
void read(void);
void speed_update(void);
void phamp(void);
void speed_init(void);
void speed_sample(void);
void speed_update(void);
void phamp(void);
void pfail(void);
void boost(void);
void prog_1(void);
void prog_1c(void);
void prog_1d(void);
void prog_2(void);
void prog_3(void);
int u_time(void);
int v_time(void);
int w_time(void);
int u_czero(void);
int v_czero(void);
int w_czero(void);
int cross(void);
void commu(void); /* commutates to next thyristor */
float cal34(void); /* returns expressions 3 + 4 */
void back(void); /* directs background programs */
void x_back(void); /* does background calculations */
float backcalc(void);
void calc(void);

/* Dummy status register. Assembler files must be modified manually to
access the true status register, ST, or asm statement must be used. */
/* bit structure */
typedef struct {
    unsigned int c    :1; /* carry flag */
    unsigned int v    :1; /* overflow flag */
    unsigned int z    :1; /* zero flag */

```

```

    unsigned int n    :1; /* negative flag */
    unsigned int uf   :1; /* floating-point underflow flag */
    unsigned int lv   :1; /* latched overflow flag */
    unsigned int luf  :1; /* latched floating point underflow flag */
    unsigned int ovm  :1; /* integer overflow mode flag */
    unsigned int rm   :1; /* repeat mode flag */
    unsigned int      :1;
    unsigned int cf   :1; /* cache freeze */
    unsigned int ce   :1; /* cache enable */
    unsigned int cc   :1; /* cache clear */
    unsigned int gie  :1; /* globle interrupt enable */
} status_bits;
/* declaration
extern volatile status_bits status;*/
/* masks */
    asm("GIE .set 02000H");

/* Useful structures */
typedef struct {
    unsigned int d0 :1;
    unsigned int d1 :1;
    unsigned int d2 :1;
    unsigned int d3 :1;
} four_bit;
typedef union {
    four_bit b;
    int      i;
} bit_int;
typedef struct {
    unsigned int stop :1;
    unsigned int pf  :1; /* phase fault - used in pfail() */
} flags_bit;
typedef union {
    flags_bit b;
    int      i;
} flags_un;

#if defined(mainprog)

/* Debug output addresses */
extern int debug_0[];
int *debug = debug_0;

/* Data output addresses */
extern int data_0[];
int *data = data_0;

/* Sine table access. The sine table has an amplitude of 1 and a
period of 300 and is in floating point. It's length is 1.5 periods. It
is accessed in a program as the array sine[0..450] */
extern float sine_0[];
float *sine = sine_0;

/* Flux table access. The flux table amplitude is set for 1Vrms L-N
output at 50Hz and is in floating point. It's length is 2 periods. It
is accessed in a program as the array flux_ref[0..600] */
extern float flux_ref_0[];
float *flux_ref = flux_ref_0;

```

```

/* Mains integral table. It's length is 1 period, from 15 to 105. It
is accessed in a program as the array itab[0..90]. */
extern float itab_0[];
float *itab = itab_0;

/* Mains double integral table. It's length is 1 period, from 15 to
105. It is accessed in a program as the array iitab[0..90]. */
extern float iitab_0[];
float *iitab = iitab_0;

/* Mains voltage table. It's range is from -15 to +15 of input zero
crossing. It is accessed in a program as the array mainsv[0..30]. The
values are the actual input phase voltages expected for a 415Vrms line
to line supply. */
extern float mainsv_0[];
float *mainsv = mainsv_0;

/* Output voltage table. It's range is from 0 to 150 of output phase.
It is accessed in a program as the array outv[0..60]. Its values are
output volts/(speed*motor_volts). */
extern float outv_0[];
float *outv = outv_0;

/* ACTEL chip access */
extern volatile unsigned int actel_0[]; /* this is the actel address
block starting at address actel_0 defined in names.asm */
volatile unsigned int *actel = actel_0; /* set up actel[] as the
array of actel addresses */
extern volatile four_bit actel_0b[]; /* this is the bit structured
version */
volatile four_bit *actel_bit = actel_0b;
/*
For Writing:
Address: Description:          Data:
0      U output              D0:bank, D1:R, D2:S, D3:T
1      V output              ditto
2      W output              ditto
3      serial out            D0:data output to serial devices
4      serial clock          D0:clock to serial devices
5      serial device select  00:A/D, 01:EEPROM, 10:LED driver

For Reading:
Address: Description:          Data:
0      U v/f counter         D0,D1,D2,D3: counter outputs
1      V v/f counter         ditto
2      W v/f counter         ditto
3      serial in             D0:data input from serial devices
4      zero current detect   D0:U, D1:V, D2:W
5      mains polarity in     D0:R-T, D1:T-S, D2:S-R
6      control in            D0:F/~R, D1:STOP/~START, D2:SET
7      speed v/f counter     D0,D1,D2,D3: counter outputs
8      tacho counter         D0,D1,D2,D3: counter outputs
9      direction             D0:speed direction, D1:tacho direction
10     general purpose       D0:PRA, D1:PRB, D2:DCLK, D3:SDI
*/

/* Global variables */
int accel = 87; /* accel rate. 2^16/4,500 = 1Hz/s/s */
float motor_volts = 420.0; /* line to neutral motor voltage at 50Hz
*/

```

```

    unsigned int phase_in = SAMP-1; /* sample number in this mains
period */
    unsigned int stime_period = 0; /* period of previous sample */
    unsigned int third_time; /* time of last 1/3 cycle in timer counts
*/
    unsigned int third_count; /* tally of timer counts for updating
third_time */
    unsigned int tricount = 0; /* modular 3 sample counter. Used in
prog_2 to run only one c_zero routine per sample. Incremented in
sample(). */
    float u_offset; /* v/f counts per 2^24 timer counts for zero output
on u phase */
    float v_offset; /* v/f counts per 2^24 timer counts for zero output
on v phase */
    float w_offset; /* v/f counts per 2^24 timer counts for zero output
on w phase */
    float flux[3] = {0.0,0.0,0.0}; /* u,v,w flux registers */
    float centre_flux = 0; /* average of above 3 fluxes. Used in czero
routine. */
    float fluxp[3] = {0.0,0.0,0.0}; /* flux[] at start of previous
sample */
    int speed_old = 0; /* present speed */
    int speed_new = 0; /* next speed */
    int speed_rem = 0; /* remainder register for out_ph[0] update */
    int speed_count; /* previous reading of speed v/f counter */
    int speed_tot = 0; /* total speed v/f count */
    int phaselag_old = 0; /* present output phase lag due to boost */
    int phaselag_new = 0; /* next */
    float amplitude_old = 0; /* present output amplitude */
    float amplitude_new = 0; /* next */
    float mboost_old = 0.379; /* present M component of boost. Set to
rms boost volts X 1.514/SAMP. */
    float mboost_new = 0.379; /* next */
    float rboost_old = 0; /* present R component of boost. Set to rms
boost volts X 1.514/SAMP. */
    float rboost_new = 0; /* next */
    int out_ph[3] = {0,100,200}; /* U,V,W output and reference phase, 0
to 299 */
    int fault_mode = 0; /* indicates type of fault when "fault" routine
is
        entered
        0: no fault
        1:
        2: phase fail
        3: prog_point[0] incorrect
        4: prog_point[1] incorrect
        5: prog_point[2] incorrect
*/
    int prog_point[3] = {1,1,1}; /* pointers to appropriate trigger
programs for u,v,w phases */
    float out_dinteg[3] = {0,0,0}; /* double integrals of u,v,w output
voltages in trigger period */
    float ifluxref[3] = {0,0,0}; /* integral of reference flux from
start of period. Used for bank crossover. Adjusted in prog_1,prog_2.
*/
    float fluxerr[3] = {0,0,0}; /* flux error at start of period. Used
for bank change over. */

```

```

float sign_c[3] = {0,0,0}; /* a sign term set to + or - 1 and is
used in prog_2 to modify the main algorithm during multiple bank
change overs. */
float x_boost[3] = {0,0,0}; /* u,v,w boost voltages at current
sample */
float boost_integ[3] = {0,0,0}; /* integrals of u,v,w boost voltages
in trigger period */
float precalc[3] = {0,0,0}; /* value calculated by background
calculations */
float k[3]; /* value of k(t2-t1) for u,v & w phases for current
period. Calculated in cross. */
flags_un flags; /* miscellaneous flags */
int wflag[3] = {1,1,1}; /* flags to indicate u,v,or w background
calculations waiting to run */
int rflag[3] = {0,0,0}; /* flags to indicate u,v,or w background
calculations are running */
int t_end[3]; /* endstop times for current period */
bit_int out_port[3]; /* image of u,v,w output ports to actel */
/* out_port[].b.d0 : bank direction, 1: positive */
/* out_port[].b.d1 : R thyristor 1: on */
/* out_port[].b.d2 : S thyristor 1: on */
/* out_port[].b.d3 : T thyristor 1: on */
int thyr[3] = {0,0,0}; /* thyristor on at end of period for u,v,w
phase
0: R, 1: S, 2: T */
int oph; /* pointer for phase data selection in an array for
foreground programs. 0,1,2 for u,v,w phases. */
int b_t1; /* start time of trigger period. Used by backcalc,
adjusted in cross */
int b_t2; /* end time of trigger period. Used by backcalc & cal34,
adjusted in cross */
int t2[3]; /* value of b_t2 returned from background calculations
for u,v,w phases */
#else
extern float sine_0[],*sine,flux_ref_0[],*flux_ref,itab_0[],*itab;
extern float iitab_0[],*iitab,mainsv_0[],*mainsv,outv_0[],*outv;
extern float motor_volts,u_offset,v_offset,w_offset;
extern float
flux[3],centre_flux,fluxp[3],amplitude_old,amplitude_new;
extern float mboost_old,mboost_new,rboost_old,rboost_new;
extern float
out_dinteg[3],ifluxref[3],fluxerr[3],sign_c[3],x_boost[3];
extern float boost_integ[3];
extern float precalc[3],k[3];
extern int debug_0[],*debug,data_0[],*data,accel;
extern int speed_old,speed_new,speed_rem,speed_count,speed_tot;
extern int phaselag_old,phaselag_new,out_ph[3],fault_mode;
extern int prog_point[3],wflag[3],rflag[3];
extern int t_end[3],thyr[3];
extern int oph,b_t1,b_t2,t2[3];
extern unsigned int
phase_in,stime_period,third_time,third_count,tricount;
extern volatile unsigned int actel_0[],*actel;
extern volatile four_bit actel_0b[],*actel_bit;
extern flags_un flags;
extern bit_int out_port[3];
#endif

/* Turn listing on */
asm(" .list");

```



```

/* MAIN */

/* starting and background program for cyclo3 */

#define mainprog
#include "cyclo3.h"

void main ()
{
/*initiate flags */
  flags.i = 0;
/*initiate outputs: negative bank, all thyristors off */
  out_port[0].i = 0;
  actel[0] = out_port[0].i;
  out_port[1].i = 0;
  actel[1] = out_port[1].i;
  out_port[2].i = 0;
  actel[2] = out_port[2].i;
/*initialise flux registers to stop start-up transient */
  flux[0] = motor_volts*flux_ref[100+out_ph[0]];
  flux[1] = motor_volts*flux_ref[100+out_ph[1]];
  flux[2] = motor_volts*flux_ref[100+out_ph[2]];
/*other jobs */
  pll_init();
  read_init();
  pll_start();
  while (1);
  {;}
}

```

```

/* BACK.C */

/* Directing routine for background calculations. */

#include "cyclo3.h"

int op; /* array pointer to indicate output phase: 0,1,2 for u,v,w
*/
/*DEBUG */
int test;

void back(void)
{
/*test for calculation not in progress: test if ur, vr and wr clear */
if((rflag[0]==0)&&(rflag[1]==0)&&(rflag[2]==0))
{
/*DEBUG */
test = 0;
/*calculation not in progress (0,1) */
/*test if program calculations waiting: test if uw, vw, or ww set */
if((wflag[0]!=0)|| (wflag[1]!=0)|| (wflag[2]!=0))
{
/*calculations waiting (0,1) */
/*test if u program waiting: test if uw set */
if(wflag[0] != 0)
{
/*u program waiting (0,1)# */
/*set ur flag */
rflag[0] = 1;
/*clear uw flag */
wflag[0] = 0;
/*start background program for u phase */
op = 0;
x_back();
} else
{
/*u program not waiting #(0,1) */
/*test if v program waiting */
if(wflag[1] != 0)
{
/*v program waiting (0,1)# */
/*set vr flag */
rflag[1] = 1;
/*clear vw flag */
wflag[1] = 0;
/*start background program for v phase */
op = 1;
x_back();
} else
{
/*w program waiting #(0,1) */
/*set wr flag */
rflag[2] = 1;
/*clear ww flag */
wflag[2] = 0;
/*start background program for w phase */
op = 2;
x_back();
}
}
}
}
}

```

```
    }  
  }  
  /*DEBUG */  
  test = 0;  
}
```

```

/* BACKCALC.C */

/* This routine does the algorithm pre-calculations [terms 1 + 2] in
the background at the start of each trigger period. The total is the
return value. */

#include "cyclo3.h"

extern int op; /* output phase pointer - see X_BACK.DOC */
extern int i_back[3][5]; /* variables at period start. See PROG_1
module */
extern float f_back[3][7]; /* ditto */

float total; /* Running total of background calculations. Stored in
precalc[] at end */
int _t1; /* b_t1, giving positive local time for easy table look-ups
*/
int _t2; /* b_t2 */
float bank; /* Set to +1 for +ve bank -1 for -ve bank */
int trans; /* Transition time for speed change */
float bst_integ; /* integral of boost voltage in trigger period for
background calculations */
float integ; /* double integral of output voltage in trigger period
for background calculations */
float b_flux; /* last flux found from flux look-up table */
int ph_out; /* output phase at current step of numerical integration
*/
float mboost; /* quadrature component of boost */
float rboost; /* in phase component of boost */
int b_speed; /* speed */
int b_speed_rem; /* speed remainder */
int b_k; /*holds k(_t2-_t1) */

float backcalc(void)
/* Called from x_back */
{

/*find _t1 & _t2 */
_t1 = b_t1;
_t2 = b_t2;
/*set bank to correct value: 1 for +ve bank, -1 for -ve bank, and set
trans to correct value, SAMP/6 for + bank, SAMP/3 for - bank */
bank = +1.0;
trans = SAMP/6;
if (out_port[op].b.d0 == 0)
{
bank = -1.0;
trans = SAMP/3;
}
/*calculate K(_t2-_t1) */
b_k = 0.5*( _t2 - _t1);
/*calculate 1(a): look up iitab table at _t2 and store in total */
total = iitab[_t2];
/*calculate & add 1(c): look up itab table, multiply by K(_t2-_t1) and
add to total */
total += b_k*itab[_t2];
/*multiply total by bank to adjust for bank direction */
total = bank*total;
/*calculate 2 and add to total */
total -= f_back[op][0]*b_k;
}

```

```

/*initialise bst_integ */
    bst_integ = f_back[op][5];
/*initialise ph_out to output phase at _t1 */
    ph_out = i_back[op][3];
/*put flux reading at ph_out in b_flux */
    b_flux = motor_volts*flux_ref[100+ph_out];
/*calculate second part of 1(d) and add to total */
    total += b_k*b_flux;
/*set parameters to calculate 1(b) + 1(e) */
    mboost = f_back[op][1];
    rboost = f_back[op][3];
    b_speed = i_back[op][0];
    b_speed_rem = i_back[op][2];
/*test if there is a transition in the period */
    if ((trans - _t1) > 0)
    {
/*transition (0,1)# */
        /*calculate 1(b) + 1(e) up to transition, adding to total */
        calc();
        /*abandon program if w & r flags set */
        if ((rflag[op]!=0)&&(wflag[op]!=0)) goto abandon;
        /*set parameters to calculate 1(b) + 1(e) from transition */
        mboost = f_back[op][2];
        rboost = f_back[op][4];
        b_speed = i_back[op][1];
        trans = _t2;
        /*calculate 1(b) + 1(e) from transition */
        calc();
        /*abandon program if w & r flags set */
        if ((rflag[op]!=0)&&(wflag[op]!=0)) goto abandon;
    } else
    {
/*no transition #(0,1) */
        /*set parameters to calculate 1(b) + 1(e) */
        trans = _t2;
        /*calculate 1(b) + 1(e) */
        calc();
        /*abandon program if w & r flags set */
        if ((rflag[op]!=0)&&(wflag[op]!=0)) goto abandon;
    }
/*calculate 1(d), first part & add to total */
    total -= b_k*b_flux;
/*calculate 1(f) and add to total */
    total -= b_k*bst_integ;
abandon:
/*return total */
    return total;
}

void calc(void)
/* Calculates 1(b) + 1(e) using numerical integration */
{
/* local variables declared to improve compilation efficiency */
float m_volts; /* motor_volts */
float tot; /* total */
int t1_; /* _t1 */
int sp_rem; /* b_speed_rem */
int ph_; /* ph_out */

```

```

float flx; /* b_flux */
float bst_int; /* bst_integ */

/*load local variables */
  m_volts = motor_volts;
  tot = total;
  t1_ = _t1;
  sp_rem = b_speed_rem;
  ph_ = ph_out;
  flx = b_flux;
  bst_int = bst_integ;
/*subtract half of flx and bst_int from tot */
  tot -= 0.5*(flx + bst_int);
/*test if t1_ not = trans-1 */
  if (t1_ != trans-1)
  {
/*t1_ not = trans-1 (0,1) */
  /*loop till t1_ = trans-1 (1,?) */
  do {
    /*increment t1_ */
    ++t1_;
    /*find next ph_ */
    /*add b_speed to sp_rem*/
    sp_rem += b_speed;
    /*add sp_rem/2^16 to ph_ */
    ph_ += (sp_rem>>16);
    /*leave remainder in sp_rem */
    sp_rem = sp_rem&0x0ffff;
    /*find flx from table */
    flx = m_volts*flux_ref[100+ph_];
    /*find bst_int */
    bst_int += mboost * sine[100+ph_] + rboost * sine[100+ph_ + 75];
    /*subtract flx and bst_int from tot */
    tot -= flx + bst_int;
    /*abandon loop if w & r flags set */
    if ((rflag[op]!=0)&&(wflag[op]!=0)) t1_ = trans-1;
    /*test if t1_ = trans-1 */
    } while (t1_ < (trans-1));
  }
/*abandon program if w & r flags set */
  if ((rflag[op]!=0)&&(wflag[op]!=0)) goto abandon;
/*set t1_ to trans */
  t1_ = trans;
/*find next ph_ */
  /*add b_speed to sp_rem*/
  sp_rem += b_speed;
  /*add sp_rem/2^16 to ph_ */
  ph_ += (sp_rem>>16);
  /*leave remainder in sp_rem */
  sp_rem = sp_rem&0x0ffff;
/*find flx from table */
  flx = m_volts*flux_ref[100+ph_];
/*find bst_int */
  bst_int += mboost * sine[100+ph_] + rboost * sine[100+ph_ + 75];
/*subtract half of flx from tot_f */
  tot -= 0.5*(flx + bst_int);
/*restore external variables */
  total = tot;
  _t1 = t1_;
  b_speed_rem = sp_rem;

```

```
    ph_out = ph_;  
    b_flux = flx;  
    bst_integ = bst_int;  
abandon: ;  
}
```

```
/* BOOST.C */
```

```
/* This routine subtracts the integral of the boost reference voltages  
from the measured integral of the output voltages. The boost reference  
voltage is obtained from looking up a sine table of amplitude one at  
the appropriate phase and scaling this by the scale factor mboost_old  
for the quadrature component or rboost_old for the in-phase component.  
A scale factor of 8.4106E-3 gives a boost voltage of 1Vrms line to  
neutral. */
```

```
{  
  
/*compensate U phase */  
    x_boost[0] = mboost_old * sine[100+out_ph[0]] + rboost_old *  
sine[100+out_ph[0] + 75];  
    flux[0] -= x_boost[0];  
/*compensate V phase */  
    x_boost[1] = mboost_old * sine[100+out_ph[1]] + rboost_old *  
sine[100+out_ph[1] + 75];  
    flux[1] -= x_boost[1];  
/*compensate W phase */  
    x_boost[2] = mboost_old * sine[100+out_ph[2]] + rboost_old *  
sine[100+out_ph[2] + 75];  
    flux[2] -= x_boost[2];  
}
```

```

/* CAL34.C */

/* Contains the program cal34() which calculates expressions 3+4 of
the main algorithm for each phase [see p.83, original research notes]
and returns the result */

#include "cyclo3.h"

extern int phase_time; /* phase time: starts at 0 at earliest
possible start of period. Defined in PROG_2. */
extern float phase_sign; /* set to 1 for + bank, -1 for - bank. Used
for table correction. Defined in PROG_2. */

float cal34(void)
{
    float result;
    float t2t;
    int time;
    float fluxn; /* estimate of flux at start of next sample */
    float fluxd; /* flux change since last sample */

    /*calculate estimate of flux at start of next sample */
    fluxd = flux[oph] - fluxp[oph];
    fluxn = flux[oph] + fluxd;
    /*calculate estimate of (3a + 3c) + 4a + 4b + 4e + 4f - (3b + 4c + 4d)
at start of next sample interval*/
    time = phase_time + 1;
    t2t = (float) (t2[oph] - time);
    result = out_dinteg[oph] + boost_integ[oph]/2.0 + fluxn/2.0 +
t2t*fluxn + k[oph]*fluxn + t2t*boost_integ[oph] \
+ k[oph]*boost_integ[oph] - phase_sign*(iitab[time] + t2t*itab[time]
+ k[oph]*itab[time]);
    /*subtract half of maximum error estimate due to sample delay */
    result -= 0.5*t2t*(-fluxd + phase_sign*(itab[time] -
itab[phase_time]));

    return result;
}

```

```

/* COMMU.C */

/* This program commutates the u, v or w outputs to the next
thyristor. The thyristor in the phase is turned off now and the next
thyristor is turned on at the start of the next sample. */

/* Called from prog_2.c */

#include "cyclo3.h"

void commu(void)
{
/*turn off thyristors */
/*turn off outputs in out_port[] */
out_port[oph].i &= 0x1;
/*send out_port[] to output */
actel[oph] = out_port[oph].i;
/*test thyr[oph] = 1 */
if(thyr[oph] == 1)
{
/*thyr[oph] = 1 (0,1)# */
/*turn on THSX in out_port[] */
out_port[oph].b.d2 = 1;
/*set thyr[oph] to 2 */
thyr[oph] = 2;
} else
{
/*thyr[oph] <> 1 #(0,1) */
/*test thyr[oph] = 2 */
if(thyr[oph] == 2)
{
/*thyr[oph] = 2 (0,1)# */
/*turn on THTX in out_port[].i */
out_port[oph].b.d3 = 1;
/*set thyr[oph] to 0 */
thyr[oph] = 0;
} else
{
/*thyr[oph] = 0 #(0,1) */
/*turn on THRX in out_port[].i */
out_port[oph].b.d1 = 1;
/*set thyr[oph] to 1 */
thyr[oph] = 1;
}
}
}
}

```

```

/* CROSS.C */

/* Calculates end of period, t2. This is the return parameter. For
details on how it works, see pages 73 & 74 of research notes. */

/* Called from x_back */

#include "cyclo3.h"

extern int op; /* output phase pointer - see X_BACK.DOC */
extern int i_back[3][5]; /* variables at period start. See PROG_1
module */

int cross(void)
{
    float tempf; /* temporary register */
    int sign; /* set to +1 or -1 depending on the output bank and phase
*/
    int time; /* phase time: starts at -5/12*SAMP earliest possible
start of period. Is equal to 0 at the zero crossing of the phase to be
triggered in this period. */
    int trans; /* transition time for speed setting */
    int po0; /* output phase at time = 0 */
    int pi1; /* time at 1st point of binary iteration */
    int pi2; /* time at 2nd point of binary iteration */
    int pi3; /* time at intermediate point of binary iteration */
    int speed; /* speed at time = 0 */
    float vi3; /* input phase voltage at time pi3. Assumes a 415Vrms
line to line supply */
    float vo3; /* ouput phase voltage/(speed*motor_volts) at time pi3.
Normalised to match vi3 */
    /*test bank direction */
    if (out_port[op].b.d0 != 0)
    {
        /*bank is positive (0,1)# */
        /*convert phase_in at period start to phase time */
        time = i_back[op][4] + SAMP/12 - thyr[op]*SAMPT;
        /*set sign to +1 */
        sign = 1;
        /*set trans to -SAMP/4 */
        trans = -SAMP/4;
    }
    else
    {
        /*bank is negative #(0,1) */
        /*convert phase_in at period start to phase time */
        time = i_back[op][4] - SAMP*5/12 - thyr[op]*SAMPT;
        /*set sign to -1 */
        sign = -1;
        /*set trans to -SAMP/12 */
        trans = -SAMP/12;
    }
    /*adjust time to range -SAMP*5/12 to SAMP*7/12-1 */
    /*test if time < -SAMP*5/12 */
    if (time < -SAMP*5/12)
    {
        /*time < -SAMP*5/12 (0,1)# */
        /*add SAMP */
        time += SAMP;
    } else

```

```

    {
/*time >= -SAMP*5/12 #(0,1) */
/*test if time >= SAMP*7/12 */
    if (time >= SAMP*7/12)
    {
/*time >= SAMP*7/12 (0,1) */
/*subtract SAMP */
        time -= SAMP;
    }
    }
/*store time + SAMP*5/12 in b_t1 for background calculations */
    b_t1 = time + SAMP*5/12;
/*calculate speed and output phase at time = 0. Put output phase in
po0 */
/*test if there is a transition between the period start time and
time = 0 */
    if ((trans - time) > 0)
    {
/*transition (0,1)# */
/*calculate output phase, po0 */
        po0 = i_back[op][3] + ((i_back[op][0]*(trans - time)\
+ i_back[op][1]*(-trans) +32768)>>16);
/*set speed to speed_new */
        speed = i_back[op][1];
    } else
    {
/*no transition #(0,1) */
/*calculate output phase, po0 */
        po0 = i_back[op][3] + ((i_back[op][0]*(-time) +32768)>>16);
/*set speed to speed_old */
        speed = i_back[op][0];
    }
/*test if po0 >= 300 */
    if (po0 >= 300)
    {
/*po0 >= 300 (0,1)# */
/*po0 = po0 - 300 */
        po0 -= 300;
    } else
    {
/*po0 < 300 #(0,1) */
/*test if po0 < 0 */
        if (po0 < 0)
        {
/*po0 < 0 (0,1) */
/*po0 = po0 + 300 */
            po0 += 300;
        }
    }
}
/*initialise binary iteration */
/*set pi1 to -SAMP/12, pi2 to SAMP/20. This is set early to provide
endstop time. */
    pi1 = -SAMP/12;
    pi2 = SAMP/20;
/*pi3 = (pi1 + pi2 + 1)>>1 */
    pi3 = (pi1 + pi2 + 1)>>1;
/*tempf = speed*motor_volts*sign */
    tempf = ((float) (speed*(motor_volts*sign)));
/*abandon program if w & r flags set */
    if ((rflag[op]!=0)&&(wflag[op]!=0)) goto abandon;

```

```

/*binary iteration to find end of period time (1,?) */
do
{
/*find vi3-vo3 and test if (vi3-vo3) >= 0 */
vi3 = mainsv[pi3+SAMP/12];
vo3 = tempf*outv[30 + po0 + ((pi3*speed + 32768)>>16)];
if ((vi3-vo3) >= 0)
{
/*(vi3-vo3) >= 0 (0,1)# */
/*pi1 = pi3 */
pi1 = pi3;
} else
{
/*(vi3-vo3) < 0 #(0,1) */
/*pi2 = pi3 */
pi2 = pi3;
}
/*pi3 = (pi1 + pi2 + 1)>>1 */
pi3 = (pi1 + pi2 + 1)>>1;
/*repeat loop if pi3 != pi2 */
} while (pi3 != pi2);
/*store pi3 + SAMP*5/12 in b_t2 for background & foreground
calculations */
b_t2 = pi3 + SAMP*5/12;
/*convert pi3 to phase_in time */
if (out_port[op].b.d0 != 0)
time = pi3 - SAMP/12 + thyr[op]*SAMPT;
else
time = pi3 -SAMP*7/12 + thyr[op]*SAMPT;
if (time < 0) time += SAMP;
/*
debug[0] = po0;
debug[1] = sign;
debug[2] = pi3;
debug[3] = thyr[op];
debug[4] = time;
debug[5] = op;*/
abandon:
/*return this value */
return time;
}

```

```

/* CZERO.C */

/* This program checks for a true current zero (two successive zero
current readings). If so, the thyristors are turned off and a check of
the flux error sign is done to see if a bank cross-over is required.
If this is the case, the program then carries out the bank cross-over.
The program returns a code to the calling program indicating its
status. */

/* The program is called from prog_2.c program. */

#include "cyclo3.h"

extern float flux_error;
int temp_out[3]; /* stores original output pattern */
int flag_cz[3] = {0,0,0}; /* 1: Indicates zero current was detected
last sample. 2: indicates zero current & thyristors off. Reset in
prog_1. */

int czero(void)
{
    int return_code; /* 0: no bank change
                      1: current is zero & possible bank change next
sample
                      2: bank change */
    /*set return code to 0 */
    return_code = 0;
    /*test if flag_cz = 0 */
    if(flag_cz[oph] == 0)
    {
        /*flag_cz = 0 (0,1)# */
        /*test for current zero: test bit oph of actel[4] */
        if((actel[4] & (1 << oph)) != 0)
        {
            /*current zero (0,1) */
            /*store output pattern in temp_out[] */
            temp_out[oph] = out_port[oph].i;
            /*turn off thyristors */
            out_port[oph].i &= 0x1;
            actel[oph] = out_port[oph].i;
            /*set flag_cz[oph] = 1 */
            flag_cz[oph] = 1;
        }
    } else
    {
        /*flag_cz[] != 0 #(0,1) */
        /*test for flag_cz[] = 2 or current zero again */
        if(flag_cz[oph] == 2 || (actel[4] & (1 << oph)) != 0)
        {
            /*flag_cz[] = 2 or current zero again (0,1)# */
            /*set flag_cz[oph] to 2 */
            flag_cz[oph] = 2;
            /*set return_code to 1 */
            return_code = 1;
            /*test bank direction for positive */
            if(out_port[oph].b.d0 != 0)
            {
                /*positive bank (0,1)# */
                /*test flux_error > 0 */
                if(flux_error > 0.1)

```

```

    {
    /*flux_error > 0 (0,1) */
    /*change bank */
    out_port[oph].i ^= 0x1;
    actel[oph] = out_port[oph].i;
    /*set return_code to 2 to indicate bank change [TEMP1] */
    return_code = 2;
    /*set thyr[oph] to 0 for phase_in<SAMPT, 1 for
SAMPT<=phase_in<2*SAMPT, 2 for 2*SAMPT<=phase_in */
    if(phase_in < SAMPT) thyr[oph] = 0;
    else if(phase_in < 2*SAMPT) thyr[oph] = 1;
    else thyr[oph] = 2;
    }
} else
{
/*negative bank #(0,1) */
/*test flux_error < 0 */
if(flux_error < -0.1)
{
/*flux_error < 0 (0,1) */
/*change bank */
out_port[oph].i ^= 0x1;
actel[oph] = out_port[oph].i;
/*set return_code to 2 to indicate bank change [TEMP1] */
return_code = 2;
/*set thyr[oph] to 1 for phase_in<SAMP/6, 2 for
SAMP/6<=phase_in<SAMP/2, 0 for SAMP/2<=phase_in<SAMP*5/6, 1 for
SAMP*5/6<=phase_in */
if(phase_in < SAMP/6) thyr[oph] = 1;
else if(phase_in < SAMP/2) thyr[oph] = 2;
else if(phase_in < SAMP*5/6) thyr[oph] = 0;
else thyr[oph] = 1;
}
}
} else
{
/*flag_cz[] != 2 and current not zero again #(0,1) */
/*set flag_cz[] to 0 */
flag_cz[oph] = 0;
/*turn thyristors back on */
out_port[oph].i = temp_out[oph];
actel[oph] = out_port[oph].i;
/*indicate no bank change: return_code = 0*/
return_code = 0;
}
}
/*return return_code */
return return_code;
}

```

```
/* FAULT.C */

#include "cyclo3.h"

void fault(void)
/* stops processor and indicates faults */
{
/*disable interrupts */
asm(" ANDN GIE,ST ;status.gie = 0");
/*turn off outputs */
actel[0] = 0;
actel[1] = 0;
actel[2] = 0;
out_port[0].i = 0;
out_port[1].i = 0;
out_port[2].i = 0;
/*infinite loop */
for (;;)
}
```

```

/* PARAM.C */

#include "cyclo3.h"

/* Variables: */

void parameter_update(void)
/* Updates all parameters that depend on time. Also calls speed
routines to determine output speed */
/* Called from sample() in sample.c */
{
/*test for input transition, phase_in = 0,SAMPT or 2*SAMPT */
/*multiply by 2**16/SAMPT+e = (65536+SAMPT)/SAMPT */
/*test lower word for < 2**16/SAMPT */
if(((phase_in * ((65536+SAMPT)/SAMPT)) & 0x0ffff) < 32220/SAMPT)
{
/*input transition (0,1) */
/*update time for last 1/3 period, third_time, then clear
third_time. third_count is incremented by the previous sample period
every sample - see pll.c. */
third_time = third_count;
third_count = 0;
/*update various parameters */
speed_old = speed_new;
phaselag_old = phaselag_new;
amplitude_old = amplitude_new;
mboost_old = mboost_new;
rboost_old = rboost_new;
/*update speed_new */
speed_update();
/*update boost register values */

/*find phaselag_new and amplitude_new, next phase lag and amplitude
of output [phamp] */
phamp();
}
/*update output angle, out_ph[0] */
/*add speed_old to speed_rem*/
speed_rem += speed_old;
/*add speed_rem/2^16 to out_ph[0] */
out_ph[0] += (speed_rem>>16);
/*leave remainder in speed_rem */
speed_rem = speed_rem&0x0ffff;
/*test for >= 300 */
if(out_ph[0] >= 300)
{
/*>= 300 (0,1)# */
/*subtract 300 */
out_ph[0] -= 300;
} else
{
/*< 300 #(0,1) */
/*test if out_ph[0] < 0 */
if (out_ph[0] < 0)
{
/*out_ph[0] < 0 (0,1) */
/*add 300 */
out_ph[0] += 300;
}
}
}
}

```

```

/*calculate out_ph[1] */
/*out_ph[1] = out_ph[0] + 100 */
  out_ph[1] = out_ph[0] + 100;
/*test for >= 300 */
  if(out_ph[1] >= 300)
  {
/*>= 300 (0,1) */
    /*out_ph[1] = out_ph[1] - 300 */
    out_ph[1] -= 300;
  }
/*calculate out_ph[2] */
/*out_ph[2] = out_ph[0] - 100 */
  out_ph[2] = out_ph[0] - 100;
/*test for < 0 */
  if(out_ph[2] < 0)
  {
/*< 0 (0,1) */
    /*out_ph[2] = out_ph[2] + 300 */
    out_ph[2] += 300;
  }
}

```

```

/* PFAIL.C */

/* This routine checks if there has been a phase failure and if so it
jumps to FAULT routine which turns off the thyristors. A two is loaded
into the accumulator to give the user an indication that PFAIL has
occurred. */
/* The R-T, T-S and R-S mains polarity inputs are first exclusive-ored
together to form a square wave of three times the mains frequency and
in phase with R-T. This square wave is checked for correct polarity at
three consecutive samples about every 1/8, 3/8, 5/8 and 7/8 of a
cycle. If the polarity is incorrect at all three samples, a fault is
registered and the fault routine called. */

{
    int input;
    int sign;
    int half_cycle;
    int phase;
    int temp;

/*exclusive or the mains polarity inputs */
    input = actel[5];
    sign = (input ^ (input>>1) ^ (input>>2)) & 0x1;
/*divide phase_in by SAMP/6, save dividend in "half_cycle", modulus in
"phase" */
    temp = phase_in * ((65536*6)/SAMP+1);
    half_cycle = temp >>16;
    phase = phase_in - half_cycle*(SAMP/6);
/*test if phase = SAMP/24 or SAMP/6-SAMP/24-2 */
    if(phase==(SAMP/24) || phase==(SAMP/6-SAMP/24-2))
    {
/*phase = SAMP/24 or SAMP/6-SAMP/24-2 (0,1)# */
/*clear fault flag flags.b.pf */
        flags.b.pf = 0;
/*test if sign = l.s.b. of half_cycle */
        if(sign == half_cycle & 0x1)
        {
/*sign = l.s.b. of half_cycle (0,1) */
/*set fault flag flags.b.pf */
            flags.b.pf = 1;
        }
    }
else
{
/*phase <> SAMP/24 or SAMP/6-SAMP/24-2 #(0,1) */
/*test if phase = SAMP/24+1 or SAMP/6-SAMP/24-1 */
    if(phase==(SAMP/24+1) || phase==(SAMP/6-SAMP/24-1))
    {
/*phase = SAMP/24+1 or SAMP/6-SAMP/24-1 (0,1)# */
/*test if phase fault flag set */
        if(flags.b.pf != 0)
        {
/*phase fault flag set (0,1) */
/*test if sign != l.s.b. of half_cycle */
            if(sign != half_cycle & 0x1)
            {
/*sign != l.s.b. of half_cycle (0,1) */
/*clear phase fault flag */
                flags.b.pf = 0;
            }
        }
    }
}
}

```



```

/* PHAMP.C */

#include "cyclo3.h"

void phamp(void)
/* Calculates the next phase lag on the output due to boost,
phaselag_new, and the next output amplitude, amplitude_new */
/* Called from parameter_update in param.c */
{
/*set phaselag_new to 0 */
    phaselag_new = 0;
/*calculate amplitude_new */
    amplitude_new = speed_new * motor_volts;
}

/* The following is a copy of the Warnier Diagram from CYCOLD:
.calculate the quadrature amplitude and put in STORE
    .STORE = MBSTNEW X BCONST/2**16
.calculate the in phase amplitude and put in TEMP
    .TEMP(q11) = VSPEED X 2**3 X VFSET/2**16
    + RBSTNEW X BCONST/2**16
.calculate sign and modulus of TEMP
    .test TEMP >= 0
    .TEMP >= 0 (0,1) #
    .TEMP1 = 1
    .TEMP < 0 (0,1)
    .TEMP1 = -1
    .negate TEMP
.test for TEMP >= STORE
.TEMP >= STORE (0,1) #
    .test TEMP = STORE
    .TEMP = STORE (0,1) #
    .PHANEW = 15
.TEMP <> STORE (0,1)
    .find tan (= STORE/TEMP) to seven bits
    .round off to 6 bits
    .table look-up of angle, put in PHANEW
    .adjust for sign
    .PHANEW = PHANEW X TEMP1
.TEMP < STORE (0,1)
    .find tan (= TEMP/STORE) to seven bits
    .round off to 6 bits
    .table look-up of angle, put in PHANEW
    .adjust for inversion
    .PHANEW = 30 - PHANEW
    .adjust for sign
    .PHANEW = PHANEW X TEMP1
.change range to 0 - 120
.find amplitude and put in AMPNEW
    .AMPNEW = sqr(TEMP**2 + STORE**2)
    .calculate TEMP**2 + STORE**2, put in TEMP
    .calculate square root
    .test TEMP for zero
    .TEMP = 0 (0,1) #
    .set AMPNEW to zero
    .TEMP <> 0 (0,1)
    .put TEMP in form TEMP X 1/TEMP1**2q12 where TEMP1 is in the
form
    2**k and TEMP is between .5 and 2,q10.
    .set TEMP1 to 2**12

```

```

        .test bits 10,9 of TEMP and adjust TEMP and TEMP1 if not
zero (0,5)
        .AND TEMP with 0600H
        .test for zero
        .zero (0,1) #
            .shift TEMP left 2
            .shift TEMP1 right 1
            .repeat loop
        .not zero (0,1)
        .end loop
    .calculate sqr of TEMP
    .change TEMP to q6 (from q10)
    .lookup table at TEMP + SQR, put in TEMP
    .calculate AMPNEWq10 (= TEMPq10 X TEMP1q12/2**12)
.jump back
*/

```

```

asm(";PLL - phase lock loop module ");

/* PLL.C */

/* This file contains all the routines for adjusting the timer0 period
with a phase locked loop to adjust the sample period.
There are 180 samples per mains cycle. This gives a period of
111.11us for a 50Hz mains frequency. Each sample period is started by
the interrupt from timer0. The period of the timer is adjusted by the
phase locked loop routine. The computer cycle time is 83ns giving a
timer input cycle time of 166ns.
The implementation is as follows:
The timer period is updated at the start of every sample period.
This is done with 3 registers: a remainder register, stime_rem; a time
adjustment register, stime_adj; and a base time register, stime_base.
stime_base is the sample period X 2^16 for 50Hz input.
stime_adj is limited to +/- stime_base/10 X 2^16 for a maximum 10%
mains frequency limit.
stime_base, stime_rem and stime_adj are added together. The result
is divided by 2^16 and loaded into the timer period register. The
modulus is stored in stime_rem.
main_count stores the total timer count since the start of a mains
period. It is updated at the start of each sample period from
stime.period, which contains the timer period of the previous sample.
At the start of a mains period, after it is updated, the value of
main_count is stored in main_period then main_count is reset to 0.
A high priority interrupt at the negative zero cross of the mains
is used to record the negative zero cross time. The timer value +
main_count are stored in main_phase. This interrupt is disabled from
when the timer is reset till after main_count is updated. The routine
for this interrupt this coded in assembler in PLL_INT.
At the initialisation procedure, stime_base is found by averaging
the count per mains cycle over 16 mains cycles. */

/* definitions, macros, includes, etc.*/

#include "cyclo3.h"
#define MAINS_MAX 126482 /* max. mains period in timer counts */
#define MAINS_MIN 114482 /* min. mains period in timer counts */

/* Dummy interrupt registers */
/* bit structure for interrupt registers */
typedef struct {
    unsigned int int0 :1;
    unsigned int int1 :1;
    unsigned int int2 :1;
    unsigned int int3 :1;
    unsigned int xint0 :1;
    unsigned int rint0 :1;
    unsigned int      :2;
    unsigned int tint0 :1;
    unsigned int tint1 :1;
} interrupt;
/* Declarations to registers
extern volatile interrupt int_enable;
extern volatile interrupt int_flags;*/
/* masks for asm inserts */
asm("INT0 .set 01H");
asm("INT1 .set 02H");
asm("INT2 .set 04H");

```

```

asm("INT3 .set 08H");
asm("XINT0 .set 010H");
asm("RINT0 .set 020H");
asm("TINT0 .set 0100H");
asm("TINT1 .set 0200H");

/* Dummy I/O register */
/* bit structure for I/O register */
typedef struct {
    unsigned int      :1;
    unsigned int ioxf0 :1;
    unsigned int outxf0 :1;
    unsigned int inxf0  :1;
    unsigned int      :1;
    unsigned int ioxf1  :1;
    unsigned int outxf1 :1;
    unsigned int inxf1  :1;
} input_output;
/* Declaration */
extern volatile input_output io;
/* masks for asm inserts */
asm("IOXF0 .set 02H");
asm("OUTXF0 .set 04H");
asm("INXF0 .set 08H");
asm("IOXF1 .set 020H");
asm("OUTXF1 .set 040H");
asm("INXF1 .set 080H");

/* timer registers */
/* bit structure for timer control registers */
typedef struct {
    unsigned int func   : 1;
    unsigned int io     : 1;
    unsigned int datout : 1;
    unsigned int datin  : 1;
    unsigned int        : 2;
    unsigned int go     : 1;
    unsigned int hld    : 1;
    unsigned int cp     : 1;
    unsigned int clksrc : 1;
    unsigned int inv    : 1;
    unsigned int tstat  : 1;
} tcon;
/* Declarations to timer 0 registers. Locations are defined in asm
file "names.doc". */
extern volatile tcon t0_control; /* control register */
extern volatile unsigned int t0_counter; /*counter register */
extern volatile unsigned int t0_period; /* period register */

/* Variables for this file */

volatile int stime_adj; /* made volatile in case pll_calculate is
interrupted while stime_adj is being updated */
volatile unsigned int stime_base = 30000*65536/SAMP*4; /* for 50Hz &
24MHz xtal.= 24M X 2^16 /4 /50 /SAMP */
volatile unsigned int stime_rem = 0;
unsigned int main_count; /* timer counts from start of current mains
cycle to start of current sample period */
unsigned int main_phase; /* sample counts from start of current
mains cycle */

```

```

    unsigned int main_period; /* total timer counts in previous mains
period */
    unsigned int main_start; /* indicates start of mains cycle when non-
zero */
    unsigned int pos_count = 0; /* counter used to verify positive mains
before interrupt 0 (for PLL_INT) is enabled */
    int phase_error1 = 0; /* previous phase error. Used in pll_calculate
*/

/* Functions */

void pll_init(void)
/* finds the mains period and initialises stime_bas. Also sets up XF0
pin as input of mains polarity */
{
    unsigned int old_time;
    unsigned int new_time;
    unsigned int error;
    unsigned int i;
    unsigned int temp;
/*initialise timer0 */
    t0_period = 0;
    --t0_period;
    t0_control.func = 1;
    t0_control.hld = 1;
    t0_control.cp = 0;
    t0_control.clksrc = 1;
    t0_control.inv = 0;
/*find time for 16 mains periods (1,?) */
    do
    {
        /*clear error */
        error = 0;
        /*wait for positive mains transition */
        pll_wait_mains_pos();
        /*start timer 0 with period set to maximum */
        t0_control.go = 1;
        /*store 0 in old_time */
        old_time = 0;
        /*wait for next cycle and check period (1,16) */
        i = 0; do { i++;
            /*wait for positive mains transition */
            pll_wait_mains_pos();
            /*read time into new_time */
            new_time = t0_counter;
            /*check period is within bounds */
            temp = new_time - old_time;
            if (temp < MAINS_MIN || temp > MAINS_MAX)
            /*period is not within bounds (0,1) */
            {
                /*indicate error and indicate last loop */
                error = 1;
                i = 16;
            }
            /*update old_time */
            old_time = new_time;
            /*check for end of loop */
        } while (i < 16);
        /*end loop if no error */
    } while (error == 1);
}

```

```

/*find stime_adj = new_time * (2^16/(SAMP*16) - stime_base */
   stime_adj = new_time/256 * (1048576/SAMP) - stime_base;
/*initialise main_count to new_time/16 */
   main_count = new_time/16;
/*initialise main_phase to main_count/2 */
   main_phase = main_count/2;
/*set up XF0 pin as input of inverse mains polarity */
   /*nil: done on reset */
}

void pll_start(void)
/* starts timer 0 and phase locked loop and runs read_start()*/
{
/*initialise timer0 with period set to (stime_bas + stime_adj)/2^16 */
   t0_period = (stime_base + stime_adj)/0x010000;
   t0_control.func = 1;
   t0_control.hld = 1;
   t0_control.cp = 0;
   t0_control.clksrc = 1;
   t0_control.inv = 0;
/*adjust main_count to correct initial value: subtract one timer
period */
   main_count -= t0_period;
/*wait for positive mains transtion to synchronise software */
   pll_wait_mains_pos();
/*initialise speed counter */
   speed_init();
/*initialise flux counters for read routines */
   read_start();
/*enable timer0 interrupts */
   asm(" OR TINT0,IE ;int_enable.tint0 = 1");
   asm(" OR TINT0,IF ;int_flags.tint0 = 1");
/*reset timer0 */
   t0_control.go = 1;
/*enable global interrupts */
   asm(" OR GIE,ST ;status.gie = 1");
}

/*DEBUG */
   extern int test;

void c_int09(void)
/* Timer 0 interrupt routine that is the main sample interrupt. Runs
180 times per mains cycle. */

{
   unsigned int temp;

/*DEBUG */
   if (test != 0) debug[16] = 0;
   test = 1;

/*update main_count, third_count and phase_in */
   main_count += t0_period;
   third_count += t0_period;
   ++phase_in;
/*test if pos_count <> 0 */
   if (pos_count != 0)
   {
/*pos_count <> 0 (0,1) */

```

```

    /*call pll_pos() to enable negative mains zero cross interrupt */
    pll_pos();
}
/*test if start of mains cycle - test if phase_in = SAMP */
if (phase_in >= SAMP)
/*start of mains cycle (0,1) */
{
    /*set main_period to main_count */
    main_period = main_count;
    /*reset main_count and phase_in to zero */
    main_count = 0;
    phase_in = 0;
    /*set main_start = TRUE */
    main_start = 1;
}
/*clear c_int09 interrupt flag */
asm(" ANDN TINT0,IF ;int_flags.tint0 = 0");
/*enable global interrupt flag */
asm(" OR GIE,ST ;status.gie = 1");
/*update previous period register, stime_period */
stime_period = t0_period;
/*update timer period register */
temp = stime_base + stime_rem + stime_adj;
stime_rem = temp%0x010000;
t0_period = temp/0x010000;
/*run sample interrupt routine */
sample();
/*test if start of mains cycle - test if main_start = TRUE */
if(main_start != 0)
/*start of mains cycle (0,1) */
{
    /*set main_start = FALSE */
    main_start = 0;
    /*call pll_calculate */
    pll_calculate();
    /*set pos_count to SAMP/9 to start positive mains search*/
    pos_count = SAMP/9;
}
}

void pll_calculate(void)
/* calculates new pll parameters once every mains cycle. Runs in the
background */
{
    int phase_error; /* present phase error */
    int temp;

    /*find phase error */
    phase_error = main_period/2 - main_phase;
    /*find required change in stime_adj and store in temp. See R&D notes
for the calculation of the coefficients */
    temp = (25363/SAMP) * phase_error - (21223/SAMP) * phase_error1;
    /*adjust stime_adj in one operation so it cannot be interrupted.
stime_adj has been declared volatile to ensure this. */
    stime_adj = stime_adj - temp;
    /*update phase_error1 */
    phase_error1 = phase_error;
}

```

```

/* PROG_1.C */

/* This module contains the trigger program prog_1 which is the first
to run in the u, v and w trigger periods for calculating trigger
instances. */

#include "cyclo3.h"
    int i_back[3][5];
    float f_back[3][7];
    extern int flag_cz[3]; /* see czero */
    extern int cz_return[3]; /* see prog_2 */
void prog_1(void)
{
    float fluxr;
    int opi;

/*load index oph into local variable opi to speed up the compiler */
    opi = oph;
/*set flag_cz[opi] to 0 */
    flag_cz[opi] = 0;
/*set cz_return[opi] to 0 */
    cz_return[opi] = 0;
/*find flux reference */
    fluxr = motor_volts*flux_ref[100+out_ph[opi]];
/*update fluxp[opi] */
    fluxp[opi] = flux[opi];
/*changes from prog_lc for start of normal period */
/*store fluxerror in fluxerr[] */
    fluxerr[opi] = flux[opi]-fluxr;
/*set integral of boost to half boost voltage */
    boost_integ[opi] = x_boost[opi]/2.0;
/*set double integral of output to half of integral*/
    out_dinteg[opi] = (flux[opi] + boost_integ[opi])/2.0;
/*set ifluxref[] to half of flux reference */
    ifluxref[opi] = fluxr/2;
/*set sign_c[] to 1 */
    sign_c[opi] = 1;

/*set w flag to start background calculations */
    wflag[opi] = 1;
/*set prog_point[opi] to 2 to run prog_2 next sample */
    prog_point[opi] = 2;
/*load variables for background program */
    i_back[opi][0] = speed_old;
    i_back[opi][1] = speed_new;
    i_back[opi][2] = speed_rem;
    i_back[opi][3] = out_ph[opi];
    i_back[opi][4] = phase_in; /*start time of period */
    f_back[opi][0] = flux[opi];
    f_back[opi][1] = mboost_old;
    f_back[opi][2] = mboost_new;
    f_back[opi][3] = rboost_old;
    f_back[opi][4] = rboost_new;
    f_back[opi][5] = boost_integ[opi];
    f_back[opi][6] = 0;
/*output data to data[opi] for data logging and adjust endpoints */
/*test bank direction */
    if(out_port[opi].b.d0 == 0)
    {
/*negative bank */

```

```

/*output 80h,phase_in in double byte */
  data[opi]= (0x08000) + (phase_in & 0x0FF);
/*adjust endstop, t_end[] */
  if (thyr[opi]==0) t_end[opi]=SAMP*5/12+SAMP/20-1;
  else if (thyr[opi]==1) t_end[opi]=SAMP*3/4+SAMP/20-1;
  else t_end[opi]=SAMP/12+SAMP/20-1;
} else
{
/*positive bank */
/*output 81h,phase_in in double byte */
  data[opi]= (0x08100) + (phase_in & 0x0FF);
/*adjust endstop, t_end[] */
  if (thyr[opi]==0) t_end[opi]=SAMP*11/12+SAMP/20-1;
  else if (thyr[opi]==1) t_end[opi]=SAMP/4+SAMP/20-1;
  else t_end[opi]=SAMP*7/12+SAMP/20-1;
}
/*output flux_ref,flux[opi]-flux_ref in double byte */
  data[opi]= (((int) fluxr)<<8) + ( ((int) (fluxerr[opi])) &0x0FF);
/* DEBUG */
/*output flux_ref,flux[opi]-flux_ref in 2 words */
/*  data[opi]= ((int) (fluxr*256));
  data[opi]= ((int) (fluxerr[opi]*256));*/
}

```

```

/* PROG_1d.C */

/* This module contains the trigger program prog_1d which runs instead
of prog_1 at the start of a bank crossover period. It is almost
identical to prog_1 except that it modifies the variables sent to the
background calculations to correct the algorithms for bank crossover.
It uses a slightly different algorithm to Prog_1c - see research
notes, p.97. */

#include "cyclo3.h"
extern int i_back[3][5];
extern float f_back[3][7];
extern int flag_cz[3]; /* see czero */
extern int cz_return[3]; /* see prog_2 */
void prog_1d(void)
{
    float cross_offset1; /* offset correction for bank cross-over to be
multiplied by the stability constant then added to precalc */
    float sign; /* set to bank sign. Used to calculate cross_offset1 */
    float fluxr;
    int opi;

/*load index oph into local variable opi to speed up the compiler */
    opi = oph;
/*set flag_cz[opi] to 0 */
    flag_cz[opi] = 0;
/*set cz_return[opi] to 0 */
    cz_return[opi] = 0;
/*find flux reference */
    fluxr = motor_volts*flux_ref[100+out_ph[opi]];
/*changes from prog_1 for bank cross over correction */
/*set integral of boost to half boost voltage */
    boost_integ[opi] = x_boost[opi]/2.0;
/*set double integral of output to half of integral*/
    out_dinteg[opi] = (flux[opi] + boost_integ[opi])/2.0;
/*test for positive bank */
    if(out_port[opi].b.d0 != 0)
    {
/*positive bank (0,1)# */
/*set sign to 1 for use in next section */
        sign = 1.0;
    }else
    {
/*negative bank #(0,1) */
/*set sign to -1 for use in next section */
        sign = -1.0;
    }
/*calculate cross_offset1 for bank change over corrections */
    cross_offset1 = flux[opi] - fluxr - sign*10.0;

/*set w flag to start background calculations */
    wflag[opi] = 1;
/*set prog_point[opi] to 2 to run prog_2 next sample */
    prog_point[opi] = 2;
/*load variables for background program */
    i_back[opi][0] = speed_old;
    i_back[opi][1] = speed_new;
    i_back[opi][2] = speed_rem;
    i_back[opi][3] = out_ph[opi];
    i_back[opi][4] = phase_in; /*start time of period */

```

```

f_back[opi][0] = flux[opi] - cross_offset1;
f_back[opi][1] = mboost_old;
f_back[opi][2] = mboost_new;
f_back[opi][3] = rboost_old;
f_back[opi][4] = rboost_new;
f_back[opi][5] = boost_integ[opi];
f_back[opi][6] = 0;
/*output data to data[opi] for data logging and adjust endpoints */
/*test bank direction */
    if(out_port[opi].b.d0 == 0)
    {
/*negative bank */
/*output 80h,phase_in in double byte */
        data[opi]= (0x0800) + (phase_in & 0xFF);
/*adjust endstop, t_end[] */
        if (thyr[opi]==0) t_end[opi]=SAMP*5/12+SAMP/20-1;
        else if (thyr[opi]==1) t_end[opi]=SAMP*3/4+SAMP/20-1;
        else t_end[opi]=SAMP/12+SAMP/20-1;
    } else
    {
/*positive bank */
/*output 81h,phase_in in double byte */
        data[opi]= (0x0810) + (phase_in & 0xFF);
/*adjust endstop, t_end[] */
        if (thyr[opi]==0) t_end[opi]=SAMP*11/12+SAMP/20-1;
        else if (thyr[opi]==1) t_end[opi]=SAMP/4+SAMP/20-1;
        else t_end[opi]=SAMP*7/12+SAMP/20-1;
    }
/*output flux_ref,flux[opi]-flux_ref in double byte */
    data[opi]= (((int) fluxr)<<8) + ( ((int) (flux[opi]-fluxr))
&0xFF);
/* DEBUG */
/*output flux_ref,flux[opi]-flux_ref in 2 words */
/* data[opi]= ((int) (fluxr*256));
   data[opi]= ((int) ((flux[opi]-fluxr)*256));*/
}

```

```

/* PROG_2.C */

/* This module contains the trigger program prog_2 which runs in the
second and subsequent samples of the u, v and w trigger period up to
when the thyristor in the phase is triggered. */

#include "cyclo3.h"

float flux_error; /* component of output flux causing ripple current
*/
int phase_time; /* phase time: starts at 0 at earliest possible
start of period. */
float phase_sign; /* set to 1 for + bank, -1 for - bank. Used for
table correction in cal34(). */
int cz_return[3]={0,0,0}; /* value returned by czero routines. Reset
in prog_1 */

void prog_2(void)
{
float calc; /* value returned by cal34() */
int flag; /* indicates a commutation for the data out section */
float temp;
int temp1; /* temporary use */
int opi;

/*load index oph into local variable opi to speed up the compiler */
opi = oph;
/*initialise flag to 0 */
flag = 0;

/*update ifluxref[] */
ifluxref[opi] += flux_ref[100+out_ph[opi]];
/*find flux_error = (output flux - reference flux - centre_flux) */
flux_error = flux[opi] - motor_volts*flux_ref[100+out_ph[opi]] -
centre_flux;
/*call CZERO : if true current zero, turn off thyristors and change
banks */
cz_return[opi] = czero();

/*test for bank change */
if(cz_return[opi] == 2)
{
/*bank change (0,1)# */
/*call prog_1d() to readjust calculations for period */
prog_1d();
} else
{
/*no bank change #(0,1) */
/*update boost integral */
boost_integ[opi] += x_boost[opi];
/*update output double integral */
out_dinteg[opi] += flux[opi] + boost_integ[opi];
/*test if end of period reached: test if phase_in = t_end[opi] */
temp1 = phase_in - t_end[opi];
if ((temp1 >= 0 && temp1 < SAMP/6) || (temp1 < SAMP/6-SAMP))
{
/*end of period reached (0,1)# */
/*commutate to next thyristor */
/*test if thyristor is off: cz_return[opi] != 0 */
if (cz_return[opi] != 0)

```

```

    {
    /*thyristor is off (0,1)* */
    /*increment thyr[opi] */
    thyr[opi] += 1;
    if (thyr[opi] == 3) thyr[opi] = 0;
    } else
    {
    /*thyristor is on *(0,1) */
    /*commutate to next thyristor turning on the thyristor */
    commu();
    }
    /*set calc to 0 for data output */
    calc = 0.0;
    /*set flag to 1 for data output */
    flag = 1;
    /*set prog_point[opi] to 1 */
    prog_point[opi] = 1;
    } else
    {
    /*end of period not reached #(0,1) */
    /*test if background calculations finished: test if w & r flags
clear */
    if ((wflag[opi]==0)&&(rflag[opi]==0))
    {
    /*background calcs finished (0,1) */
    /*test for positive bank */
    if(out_port[opi].b.d0 != 0)
    {
    /*positive bank (0,1) # */
    /*find phase_time */
    phase_time = phase_in + SAMP/2 - thyr[opi]*SAMPT;
    if (phase_time < 0) phase_time += SAMP;
    else if (phase_time >= SAMP) phase_time -= SAMP;
    /*set phase_sign to +1 */
    phase_sign = 1.0;
    /*calculate expressions 3 + 4 to start of next sample */
    calc = cal34();
    /*test precalc[] + calc + .5* error estimate >= 0 */
    if(precalc[opi] + calc >= 0.0)
    {
    /*precalc[] + calc >= 0 (0,1) # */
    /*set prog_point[opi] to 2 */
    prog_point[opi] = 2;
    } else
    {
    /*precalc[] + calc < 0 (0,1) */
    /*commutate to next thyristor */
    commu();
    /*set flag to 1 for data output */
    flag = 1;
    /*set prog_point[opi] to 3 */
    prog_point[opi] = 3;
    }
    } else
    {
    /*negative bank (0,1) */
    /*find phase_time */
    phase_time = phase_in - thyr[opi]*SAMPT;
    if (phase_time < 0) phase_time += SAMP;
    /*set phase_sign to -1 */

```

```

    phase_sign = -1.0;
    /*calculate expressions 3 + 4 to start of next sample */
    calc = cal34();
    /*test precalc[] + calc <= 0 */
    if(precalc[opi] + calc <= 0.0)
    {
        /*precalc[] + calc <= 0 (0,1) # */
        /*set prog_point[opi] to 2 */
        prog_point[opi] = 2;
    } else
    {
        /*precalc[] + calc > 0 (0,1) */
        /*commutate to next thyristor */
        commu();
        /*set flag to 1 for data output */
        flag = 1;
        /*set prog_point[opi] to 3 */
        prog_point[opi] = 3;
    }
}
}
}
/*output data to data[opi] for data logging */
/*test for commutation: test if flag = 1 */
if (flag == 1)
{
    /*commutation (0,1) */
    /*output 82h,phase_in in double byte */
    data[opi]= (0x08200) + (phase_in & 0x0FF);
    /*output precalc[opi] */
    data[opi] = precalc[opi];
    /*output calc */
    data[opi] = calc;
}
/*output flux_ref,flux[opi]-flux_ref in double byte */
temp = motor_volts*flux_ref[100+out_ph[opi]];
data[opi]= (((int) temp)<<8) + ( ((int) (flux[opi]-temp))
&0x0FF);
/* DEBUG */
/*output flux_ref,flux[opi]-flux_ref in words */
/* data[opi]= ((int) (temp*256));
data[opi]= ((int) ((flux[opi]-temp)*256));*/
}
/*update fluxp[opi] */
fluxp[opi] = flux[opi];
}

```

```

/* PROG_3.C */

/* This module contains the trigger program prog_3 which runs in the
samples of the u, v and w trigger period after the thyristor in the
phase is triggered. */

#include "cyclo3.h"

void prog_3(void)
{
    int time;
    float temp;

    /*test if this interval is end of period, t_end[oph] = phase_in */
    if(t_end[oph] == phase_in)
    {
        /*this is end (0,1)# */
        /*turn off trigger signal : disabled */
        /*prog_point[oph] = 1 */
        prog_point[oph] = 1;
    } else
    /*this is not end #(0,1) */
    /*prog_point[oph] = 3 */
    prog_point[oph] = 3;
    /*output flux_ref,flux[oph]-flux_ref in double byte */
    temp = motor_volts*flux_ref[100+out_ph[oph]];
    data[oph]= (((int) temp)<<8) + ( ((int) (flux[oph]-temp)) &0xFF);
    /* DEBUG */
    /*output flux_ref,flux[oph]-flux_ref in double byte */
    /* data[oph]= ((int) (temp*256));
    data[oph]= ((int) ((flux[oph]-temp)*256));*/
}

```

```

/* READ */

/* read() reads the U, V and W V/F counters, adjusts the readings for
offsets, then updates the flux registers, flux[0,1,2].
The offset adjustments are made using u_offset, v_offset and
w_offset, which contain the v/f counts per 2^24 timer counts for zero
output. These variables are initialised at start-up in the read_init()
routine. */

/* Called from: sample() */

#include "cyclo3.h"

/* Variables */
unsigned int u_flux_count = 0;
unsigned int v_flux_count = 0;
unsigned int w_flux_count = 0;
unsigned int u_flux_offset;
unsigned int v_flux_offset;
unsigned int w_flux_offset;
unsigned int u_flux_rem = 0;
unsigned int v_flux_rem = 0;
unsigned int w_flux_rem = 0;

void read_init(void)
/* initialises flux_offset */
{
    unsigned int total;
    unsigned int count;
    unsigned int temp1;
    unsigned int temp2;
    unsigned int j;

/*find flux offset for u phase */
/*set total to zero */
    total = 0;
/*set count to vf counter value */
    count = actel[0];
/*loop so that 2^24 * 2 machine cycles occurs between the first and
last read of actel[0] */
    for(j=0; j<5592405; j++)
    {
/*read vf counter value */
        temp1 = actel[0];
/*subtract previous count to get change then update count */
        temp2 = temp1 - count;
        temp2 &= 0x0000000F;
        count = temp1;
/*add to total */
        total += temp2;
    }
/*put total into u_flux_offset */
    u_flux_offset = total;

/*find flux offset for v phase */
/*set total to zero */
    total = 0;
/*set count to vf counter value */
    count = actel[1];

```

```

    /*loop so that 2^24 * 2 machine cycles occurs between the first and
last read of actel[1] */
    for(j=0; j<5592405; j++)
    {
        /*read vf counter value */
        temp1 = actel[1];
        /*subtract previous count to get change then update count */
        temp2 = temp1 - count;
        temp2 &= 0x0000000F;
        count = temp1;
        /*add to total */
        total += temp2;
    }
    /*put total into v_flux_offset */
    v_flux_offset = total;

/*find flux offset for w phase */
/*set total to zero */
    total = 0;
    /*set count to vf counter value */
    count = actel[2];
    /*loop so that 2^24 * 2 machine cycles occurs between the first and
last read of actel[2] */
    for(j=0; j<5592405; j++)
    {
        /*read vf counter value */
        temp1 = actel[2];
        /*subtract previous count to get change then update count */
        temp2 = temp1 - count;
        temp2 &= 0x0000000F;
        count = temp1;
        /*add to total */
        total += temp2;
    }
    /*put total into w_flux_offset */
    w_flux_offset = total;
}

void read_start(void)
/* initialises flux_count just before starting. Called from
pll_start() */
{
    u_flux_count = actel[0]-3;
    v_flux_count = actel[1]-3;
    w_flux_count = actel[2]-3;
}

void read(void)
/* reads the U, V and W V/F counters, adjusts the readings for ofsets,
then updates the flux registers, flux[0,1,2] */
{
    int flux_change;
    unsigned int u_flux_read;
    unsigned int v_flux_read;
    unsigned int w_flux_read;
    unsigned int temp;

/*read flux */
    u_flux_read = actel[0];

```

```

v_flux_read = actel[1];
w_flux_read = actel[2];

/*adjust flux for u phase */
/*extract flux value */
/*subtract previous count, flux_count, from present flux */
temp = u_flux_read - u_flux_count;
/*update previous count, flux_count, with new count */
u_flux_count = u_flux_read;
/*correct offset */
flux_change = temp & 0x0000000F;
temp = (stime_period * u_flux_offset)/256 + u_flux_rem;
flux_change -= temp/65536;
/*update remainder, flux_rem */
u_flux_rem = temp%65536;
/*calculate new motor flux */
/*add flux change to previous flux value */
flux[0] += flux_change;
/*check for negative flux overflow: check if < -FLUXMAX */
if(flux[0] < -FLUXMAX)
{
/*negative overflow (0,1)# */
/*limit to -FLUXMAX */
flux[0] = -FLUXMAX;
}
/*not negative overflow #(0,1) */
/*check for positive overflow: check if > FLUXMAX */
else if(flux[0] > FLUXMAX)
{
/*positive overflow (0,1) */
/*limit to FLUXMAX */
flux[0] = FLUXMAX;
}
}

/*adjust flux for v phase */
/*extract flux value */
/*subtract previous count, flux_count, from present flux */
temp = v_flux_read - v_flux_count;
/*update previous count, flux_count, with new count */
v_flux_count = v_flux_read;
/*correct offset */
flux_change = temp & 0x0000000F;
temp = (stime_period * v_flux_offset)/256 + v_flux_rem;
flux_change -= temp/65536;
/*update remainder, flux_rem */
v_flux_rem = temp%65536;
/*calculate new motor flux */
/*add flux change to previous flux value */
flux[1] += flux_change;
/*check for negative flux overflow: check if < -FLUXMAX */
if(flux[1] < -FLUXMAX)
{
/*negative overflow (0,1)# */
/*limit to -FLUXMAX */
flux[1] = -FLUXMAX;
}
/*not negative overflow #(0,1) */
/*check for positive overflow: check if > FLUXMAX */
else if(flux[1] > FLUXMAX)
{

```

```

        /*positive overflow (0,1) */
        /*limit to FLUXMAX */
        flux[1] = FLUXMAX;
    }

/*adjust flux for w phase */
/*extract flux value */
    /*subtract previous count, flux_count, from present flux */
    temp = w_flux_read - w_flux_count;
    /*update previous count, flux_count, with new count */
    w_flux_count = w_flux_read;
    /*correct offset */
    flux_change = temp & 0x0000000F;
    temp = (stime_period * w_flux_offset)/256 + w_flux_rem;
    flux_change -= temp/65536;
    /*update remainder, flux_rem */
    w_flux_rem = temp%65536;
/*calculate new motor flux */
    /*add flux change to previous flux value */
    flux[2] += flux_change;
    /*check for negative flux overflow: check if < -FLUXMAX */
    if(flux[2] < -FLUXMAX)
    {
        /*negative overflow (0,1)# */
        /*limit to -FLUXMAX */
        flux[2] = -FLUXMAX;
    }
    /*not negative overflow #(0,1) */
    /*check for positive overflow: check if > FLUXMAX */
    else if(flux[2] > FLUXMAX)
    {
        /*positive overflow (0,1) */
        /*limit to FLUXMAX */
        flux[2] = FLUXMAX;
    }
}

```

```

/* SAMPLE.C */

/* runs all routines that are repeated every sample instant */

#include "cyclo3.h"

void sample(void)
{
    /*update thyristor status */
    actel[0] = out_port[0].i;
    actel[1] = out_port[1].i;
    actel[2] = out_port[2].i;
    read();
    speed_sample();
#include "pfail.doc"
    parameter_update();
#include "boost.doc"

    /*increment tricount, modular 3. Used in prog_2() to run czero
routine */
    ++ tricount;
    if (tricount > 2) tricount = 0;

    centre_flux = (1/3.0)*(flux[0] + flux[1] + flux[2]);
    oph = 0;
    if (prog_point[0] == 1)
        prog_1();
    else
        if (prog_point[0] == 2)
            prog_2();
        else
            prog_3();

    oph = 1;
    if (prog_point[1] == 1)
        prog_1();
    else
        if (prog_point[1] == 2)
            prog_2();
        else
            prog_3();

    oph = 2;
    if (prog_point[2] == 1)
        prog_1();
    else
        if (prog_point[2] == 2)
            prog_2();
        else
            prog_3();

/*run background calculations */
    back();
}

```

```

/* SPEED.C */

/* Various routines for initialising and modifying speed parameters */

#include "cyclo3.h"

void speed_init(void)
/* This routine reads in the speed v/f counter and initialises
speed_count with its value. */
/* Called from pll_start() */
{
    speed_count = actel[7];
}

void speed_sample(void)
/* This routine runs at the start of each sample period. It reads in
the speed v/f counter and adds the change in the count to speed_tot.
Note that the speed v/f counter is an up/down counter. */
/* Called from sample() */
{
    int temp1;
    int temp2;

    /*read speed v/f counter */
    temp1 = actel[7];
    /*subtract previous count */
    temp2 = temp1 - speed_count;
    /*mask to 4 bits. The actel data is only 4 bits wide. */
    temp2 &= 0x0f;
    /*add to speed_tot */
    speed_tot += temp2;
    /* update speed_count */
    speed_count = temp1;
}

void speed_update(void)
/* corrects for direction and updates variable speed_new taking into
account max. acceleration. This runs every 1/3 mains cycle. */

/* Called from: param() in param.c */
{
    int speed;
    unsigned int temp;

    /*find actual speed, scaled so that v/f freq. = 1000 * output freq. */
    speed = speed_tot * ((58982+SAMP/2)/SAMP);
    /*read direction from bit 0 of actel[6] */
    temp = actel[6];
    /*negate speed if direction is reverse */
    if ((temp&0x01) == 0)
        speed = -speed;
    /*reset speed_tot to zero */
    speed_tot = 0;
    /*clip speed to SPEED_POSMAX, -SPEED_NEGMAX */
    if(speed > SPEED_POSMAX) speed = SPEED_POSMAX;
    else if(speed < -SPEED_NEGMAX) speed = -SPEED_NEGMAX;
    /*update speed_new, with limited acceleration */
    if(speed > speed_new)
    {

```

```
    speed_new += accel;
    if(speed < speed_new)
    {
        speed_new = speed;
    }
}
else
{
    speed_new -= accel;
    if(speed > speed_new)
    {
        speed_new = speed;
    }
}
}
```

```

/* TIME.C */

/* Calculates time in terms of input phase. */

#include "cyclo3.h"

int u_time(void)
/* Calculates time for output phase u */
{
    int time;
/*calculate time */
    time = phase_in + 30 - thyr[0] * 60;
    if (time < 0)
    {
        time += 180;
    }
    else
    {
        if (time >= 180)
        {
            time -= 180;
        }
    }
/*return time */
    return time;
}

int v_time(void)
/* Calculates time for output phase v */
{
    int time;
    time = phase_in + 30 - thyr[1] * 60;
    if (time < 0)
    {
        time += 180;
    }
    else
    {
        if (time >= 180)
        {
            time -= 180;
        }
    }
    return time;
}

int w_time(void)
/* Calculates time for output phase w */
{
    int time;
    time = phase_in + 30 - thyr[2] * 60;
    if (time < 0)
    {
        time += 180;
    }
    else
    {
        if (time >= 180)
        {
            time -= 180;
        }
    }
}

```

```
    }  
  }  
  return time;  
}
```

```

/* X_BACK.C */

/* Contains the program x_back(). Does the precalculations at the
start of each trigger period and stores the result in precalc[x], x =
0,1,2. */

#include "cyclo3.h"

extern int op; /* array pointer defined in BACK */
extern int b_k; /*holds k(_t2-_t1) */

void x_back(void)
{
    float b_precalc;
    int t_fin; /* end time of current period for u,v,w phases */

    /*find end time, t2 */
    t_fin = cross() ;
    /*abandon program if w & r flags set */
    if ((rflag[op]!=0)&&(wflag[op]!=0)) goto abandon;
    /*find endstop time */
    t_end[op] = t_fin-1;
    if (t_end[op] < 0) t_end[op] += SAMP;
    /*find the precalculation value of the main algorithm [terms 1 + 2] */
    b_precalc = backcalc();
    /*abandon program if w & r flags set */
    if ((rflag[op]!=0)&&(wflag[op]!=0)) goto abandon;
    /*load foreground variables from background variables */
    precalc[op] = b_precalc;
    k[op] = b_k;
    t2[op] = b_t2;
abandon:
    /*clear run flag */
    rflag[op] = 0;
}

```

```

        .title      "NAMES"
; Resolves all external memory locations for "C" program modules.

define    $MACRO name,value
        .globl    :name:
:name:    .set      :value:
        $ENDM

        define    _t0_control ,808020h
        define    _t0_counter ,808024h
        define    _t0_period  ,808028h
        define    _actel_0    ,100000h ;start address of actel memory block
        define    _debug_0    ,600000h
        define    _data_0     ,900000h

;C program start-up:
;set bus control register to full speed: 0 0000 0000 0010
LDI  02H,R0
LDP  808064H
STI  R0,@808064H
;send XF1 low to allow interrupts thru the PLD
OR  020H,IOF
ANDN 040H,IOF
;load branch to interrupt instructions
LDP  i01
LDI  @i01,R0
LDP  800000h
STI  R0,@9fc1h
LDP  i09
LDI  @i09,R0
LDP  800000h
STI  R0,@9fc9h
;jump to _c_int00
.ref _c_int00
BR   _c_int00

;Constants for C program start-up
.ref _c_int09,pll_int
i01 .word 60000000h+pll_int
i09 .word 60000000h+_c_int09

;DEBUG: define table locations
.globl _actel_0b
_actel_0b:

.end

```

```

        .title    "PLL_INT"

; Mains negative zero cross interrupt routine.
; Finds the mains phase, _main_phase.
; Derived from the compilation of the following "C" routine:
;void c_int01(void)
;{
;  if (io.inxf0 == 0)
;  {
;    main_phase = main_count + t0_counter;
;    int_enable.int0 = 0;
;  }
;  else
;    int_flags.int0 = 0;
;  status.gie = 1;
;}

;INTERRUPT SOURCE: external interrupt 0

DATA:
    .globl _main_count,_t0_counter,_main_phase

;CONSTANTS:

;ADDRESSES:
    .def pll_int

;PROGRAM:
pll_int:
;save registers
    PUSH  ST
    PUSH  R0
    PUSHF R0
;test for a valid interrupt: test if mains is negative
    TSTB  08h,IOF
    BNZ   L1
;mains is negative (0,1)#
    ;find phase, main_phase, main_phase = main_count + t0_counter
    LDI   @_main_count,R0
    BD    L2
    ADDI  @_t0_counter,R0
    STI   R0,@_main_phase
    ;disable interrupt
    ANDN  01H,IE
;*** B    L2 ;Branch occurs
L1:
;mains is not negative #(0,1)
    ;clear interrupt flag
    ANDN  01H,IF
L2:
;restore registers
    POPF  R0
    POP   R0
    POP   ST
;enable global interrupts
    OR    02000H,ST
;return
    RETI

```

.end

```

        .title  "PLL_POS"

; This is called by the sample interrupt routine cint_09 in PLL if the
; counter variable _pos_count is not zero. The routine decrements this
; counter if the mains is positive then enables the interrupt for the
; negative mains zero cross if the counter is zero.

;CALLED FROM: PLL.C

;ADDRESSES:
        .def  _pll_pos
        .ref  _pos_count

;ASSIGNMENTS:
FP .set AR3

;PROGRAM:
_pll_pos:
;initialise stack and frame pointer for "C" interface
        PUSH  FP
        LDI   SP,FP
        ADDI  1,SP
;test for positive mains
        LDI   IOF,R0
        TSTB  8,R0
        BZ    L8
;mains is positive (0,1)
        ;decrement _pos_count then test for zero
        LDI   @_pos_count,R0
        SUBI  1,R0
        STI   R0,@_pos_count
        BNZ   L1
        ;_pos_count is zero (0,1)
        ;clear int0 flag then enable int0 interrupt
        ANDN  01H,IF
        OR    01H,IE
L1:
L8:
;return to "C" calling routine
        LDI   *-FP(1),R1
        BD    R1
        LDI   *FP,FP
        NOP
        SUBI  3,SP
;*** B R1 ;BRANCH OCCURS

        .end

```

```

        .title "PLL_WAIT"

;Waits for a negative to positive transition of the mains
;Derived from the following "C" routine:
;{
;  int i;
;  do {
;    do ; while (io.inxf0 != 0);
;    for (i=1700; i > 0; --i);
;  } while (io.inxf0 != 0);
;  do ; while (io.inxf0 == 0);
;}

;CALLED FROM: PLL.C

;ADDRESSES:
        .def _pll_wait_mains_pos

;ASSIGNMENTS:
FP .set AR3

;PROGRAM:
_pll_wait_mains_pos:
;initialise stack and frame pointer for "C" interface
        PUSH  FP
        LDI   SP,FP
        ADDI  1,SP
;loop (1,.)
L5:
        ;wait for negative mains
        LDI   IOF,R0
        TSTB  8,R0
        BNZ   L5
        ;wait 1 ms
        LDI   1700,R0
        STI   R0,*+FP(1)
L6:
        LDI   *+FP(1),R0
        SUBI  1,R0
        STI   R0,*+FP(1)
        BGT   L6
        ;end loop if mains still negative
        LDI   IOF,R1
        TSTB  8,R1
        BNZ   L5
;wait for positive mains
L8:
        LDI   IOF,R0
        TSTB  8,R0
        BZ    L8
;return to "C" calling routine
        LDI   *-FP(1),R1
        BD    R1
        LDI   *FP,FP
        NOP
        SUBI  3,SP
;*** B R1 ;BRANCH OCCURS

        .end

```

```
.title "FLUXTAB"  
; Flux look-up table. Contains the floating point flux look-up table  
of 390 steps length with 300 steps per cycle.
```

```
.globl _flux_ref_0  
_flux_ref_0 .float -2.0868082E-0001,-2.1199881E-0001,-2.1522085E-  
0001,-2.1834262E-0001,-2.2135980E-0001  
.float -2.2426821E-0001,-2.2706375E-0001,-2.2974238E-0001,-  
2.3230027E-0001,-2.3473363E-0001  
.float -2.3703885E-0001,-2.3921251E-0001,-2.4125132E-0001,-  
2.4315214E-0001,-2.4491209E-0001  
.float -2.4652843E-0001,-2.4799864E-0001,-2.4932042E-0001,-  
2.5049168E-0001,-2.5151059E-0001  
.float -2.5237551E-0001,-2.5308508E-0001,-2.5363812E-0001,-  
2.5403380E-0001,-2.5427148E-0001  
.float -2.5435072E-0001,-2.5427148E-0001,-2.5403380E-0001,-  
2.5363812E-0001,-2.5308508E-0001  
.float -2.5237551E-0001,-2.5151059E-0001,-2.5049168E-0001,-  
2.4932042E-0001,-2.4799864E-0001  
.float -2.4652843E-0001,-2.4491209E-0001,-2.4315214E-0001,-  
2.4125132E-0001,-2.3921251E-0001  
.float -2.3703885E-0001,-2.3473363E-0001,-2.3230027E-0001,-  
2.2974238E-0001,-2.2706375E-0001  
.float -2.2426821E-0001,-2.2135980E-0001,-2.1834262E-0001,-  
2.1522085E-0001,-2.1199881E-0001  
.float -2.0868082E-0001,-2.0527130E-0001,-2.0177469E-0001,-  
1.9819546E-0001,-1.9453810E-0001  
.float -1.9080707E-0001,-1.8700688E-0001,-1.8314195E-0001,-  
1.7921667E-0001,-1.7523539E-0001  
.float -1.7120244E-0001,-1.6712198E-0001,-1.6299814E-0001,-  
1.5883496E-0001,-1.5463634E-0001  
.float -1.5040608E-0001,-1.4614783E-0001,-1.4186513E-0001,-  
1.3756135E-0001,-1.3323973E-0001  
.float -1.2890331E-0001,-1.2455504E-0001,-1.2019760E-0001,-  
1.1583357E-0001,-1.1146532E-0001  
.float -1.0709505E-0001,-1.0272476E-0001,-9.8356277E-0002,-  
9.3991242E-0002,-8.9631110E-0002  
.float -8.5277148E-0002,-8.0930442E-0002,-7.6591901E-0002,-  
7.2262250E-0002,-6.7942061E-0002  
.float -6.3631713E-0002,-5.9331447E-0002,-5.5041339E-0002,-  
5.0761323E-0002,-4.6491187E-0002  
.float -4.2230591E-0002,-3.7979074E-0002,-3.3736061E-0002,-  
2.9500874E-0002,-2.5272740E-0002  
.float -2.1050807E-0002,-1.6834155E-0002,-1.2621794E-0002,-  
8.4126983E-0003,-4.2057969E-0003  
.float 0.0000000E+0000, 4.2057969E-0003, 8.4126983E-0003,  
1.2621794E-0002, 1.6834155E-0002  
.float 2.1050807E-0002, 2.5272740E-0002, 2.9500874E-0002,  
3.3736061E-0002, 3.7979074E-0002  
.float 4.2230591E-0002, 4.6491187E-0002, 5.0761323E-0002,  
5.5041339E-0002, 5.9331447E-0002  
.float 6.3631713E-0002, 6.7942061E-0002, 7.2262250E-0002,  
7.6591901E-0002, 8.0930442E-0002  
.float 8.5277148E-0002, 8.9631110E-0002, 9.3991242E-0002,  
9.8356277E-0002, 1.0272476E-0001  
.float 1.0709505E-0001, 1.1146532E-0001, 1.1583357E-0001,  
1.2019760E-0001, 1.2455504E-0001  
.float 1.2890331E-0001, 1.3323973E-0001, 1.3756135E-0001,  
1.4186513E-0001, 1.4614783E-0001
```

```

.float 1.5040608E-0001, 1.5463634E-0001, 1.5883496E-0001,
1.6299814E-0001, 1.6712198E-0001
.float 1.7120244E-0001, 1.7523539E-0001, 1.7921667E-0001,
1.8314195E-0001, 1.8700688E-0001
.float 1.9080707E-0001, 1.9453810E-0001, 1.9819546E-0001,
2.0177469E-0001, 2.0527130E-0001
.float 2.0868082E-0001, 2.1199881E-0001, 2.1522085E-0001,
2.1834262E-0001, 2.2135980E-0001
.float 2.2426821E-0001, 2.2706375E-0001, 2.2974238E-0001,
2.3230027E-0001, 2.3473363E-0001
.float 2.3703885E-0001, 2.3921251E-0001, 2.4125132E-0001,
2.4315214E-0001, 2.4491209E-0001
.float 2.4652843E-0001, 2.4799864E-0001, 2.4932042E-0001,
2.5049168E-0001, 2.5151059E-0001
.float 2.5237551E-0001, 2.5308508E-0001, 2.5363812E-0001,
2.5403380E-0001, 2.5427148E-0001
.float 2.5435072E-0001, 2.5427148E-0001, 2.5403380E-0001,
2.5363812E-0001, 2.5308508E-0001
.float 2.5237551E-0001, 2.5151059E-0001, 2.5049168E-0001,
2.4932042E-0001, 2.4799864E-0001
.float 2.4652843E-0001, 2.4491209E-0001, 2.4315214E-0001,
2.4125132E-0001, 2.3921251E-0001
.float 2.3703885E-0001, 2.3473363E-0001, 2.3230027E-0001,
2.2974238E-0001, 2.2706375E-0001
.float 2.2426821E-0001, 2.2135980E-0001, 2.1834262E-0001,
2.1522085E-0001, 2.1199881E-0001
.float 2.0868082E-0001, 2.0527130E-0001, 2.0177469E-0001,
1.9819546E-0001, 1.9453810E-0001
.float 1.9080707E-0001, 1.8700688E-0001, 1.8314195E-0001,
1.7921667E-0001, 1.7523539E-0001
.float 1.7120244E-0001, 1.6712198E-0001, 1.6299814E-0001,
1.5883496E-0001, 1.5463634E-0001
.float 1.5040608E-0001, 1.4614783E-0001, 1.4186513E-0001,
1.3756135E-0001, 1.3323973E-0001
.float 1.2890331E-0001, 1.2455504E-0001, 1.2019760E-0001,
1.1583357E-0001, 1.1146532E-0001
.float 1.0709505E-0001, 1.0272476E-0001, 9.8356277E-0002,
9.3991242E-0002, 8.9631110E-0002
.float 8.5277148E-0002, 8.0930442E-0002, 7.6591901E-0002,
7.2262250E-0002, 6.7942061E-0002
.float 6.3631713E-0002, 5.9331447E-0002, 5.5041339E-0002,
5.0761323E-0002, 4.6491187E-0002
.float 4.2230591E-0002, 3.7979074E-0002, 3.3736061E-0002,
2.9500874E-0002, 2.5272740E-0002
.float 2.1050807E-0002, 1.6834155E-0002, 1.2621794E-0002,
8.4126983E-0003, 4.2057969E-0003
.float 2.9028171E-0021, -4.2057969E-0003, -8.4126983E-0003, -
1.2621794E-0002, -1.6834155E-0002
.float -2.1050807E-0002, -2.5272740E-0002, -2.9500874E-0002, -
3.3736061E-0002, -3.7979074E-0002
.float -4.2230591E-0002, -4.6491187E-0002, -5.0761323E-0002, -
5.5041339E-0002, -5.9331447E-0002
.float -6.3631713E-0002, -6.7942061E-0002, -7.2262250E-0002, -
7.6591901E-0002, -8.0930442E-0002
.float -8.5277148E-0002, -8.9631110E-0002, -9.3991242E-0002, -
9.8356277E-0002, -1.0272476E-0001
.float -1.0709505E-0001, -1.1146532E-0001, -1.1583357E-0001, -
1.2019760E-0001, -1.2455504E-0001
.float -1.2890331E-0001, -1.3323973E-0001, -1.3756135E-0001, -
1.4186513E-0001, -1.4614783E-0001

```

```

.float -1.5040608E-0001,-1.5463634E-0001,-1.5883496E-0001,-
1.6299814E-0001,-1.6712198E-0001
.float -1.7120244E-0001,-1.7523539E-0001,-1.7921667E-0001,-
1.8314195E-0001,-1.8700688E-0001
.float -1.9080707E-0001,-1.9453810E-0001,-1.9819546E-0001,-
2.0177469E-0001,-2.0527130E-0001
.float -2.0868082E-0001,-2.1199881E-0001,-2.1522085E-0001,-
2.1834262E-0001,-2.2135980E-0001
.float -2.2426821E-0001,-2.2706375E-0001,-2.2974238E-0001,-
2.3230027E-0001,-2.3473363E-0001
.float -2.3703885E-0001,-2.3921251E-0001,-2.4125132E-0001,-
2.4315214E-0001,-2.4491209E-0001
.float -2.4652843E-0001,-2.4799864E-0001,-2.4932042E-0001,-
2.5049168E-0001,-2.5151059E-0001
.float -2.5237551E-0001,-2.5308508E-0001,-2.5363812E-0001,-
2.5403380E-0001,-2.5427148E-0001
.float -2.5435072E-0001,-2.5427148E-0001,-2.5403380E-0001,-
2.5363812E-0001,-2.5308508E-0001
.float -2.5237551E-0001,-2.5151059E-0001,-2.5049168E-0001,-
2.4932042E-0001,-2.4799864E-0001
.float -2.4652843E-0001,-2.4491209E-0001,-2.4315214E-0001,-
2.4125132E-0001,-2.3921251E-0001
.float -2.3703885E-0001,-2.3473363E-0001,-2.3230027E-0001,-
2.2974238E-0001,-2.2706375E-0001
.float -2.2426821E-0001,-2.2135980E-0001,-2.1834262E-0001,-
2.1522085E-0001,-2.1199881E-0001
.float -2.0868082E-0001,-2.0527130E-0001,-2.0177469E-0001,-
1.9819546E-0001,-1.9453810E-0001
.float -1.9080707E-0001,-1.8700688E-0001,-1.8314195E-0001,-
1.7921667E-0001,-1.7523539E-0001
.float -1.7120244E-0001,-1.6712198E-0001,-1.6299814E-0001,-
1.5883496E-0001,-1.5463634E-0001
.float -1.5040608E-0001,-1.4614783E-0001,-1.4186513E-0001,-
1.3756135E-0001,-1.3323973E-0001
.float -1.2890331E-0001,-1.2455504E-0001,-1.2019760E-0001,-
1.1583357E-0001,-1.1146532E-0001
.float -1.0709505E-0001,-1.0272476E-0001,-9.8356277E-0002,-
9.3991242E-0002,-8.9631110E-0002
.float -8.5277148E-0002,-8.0930442E-0002,-7.6591901E-0002,-
7.2262250E-0002,-6.7942061E-0002
.float -6.3631713E-0002,-5.9331447E-0002,-5.5041339E-0002,-
5.0761323E-0002,-4.6491187E-0002
.float -4.2230591E-0002,-3.7979074E-0002,-3.3736061E-0002,-
2.9500874E-0002,-2.5272740E-0002
.float -2.1050807E-0002,-1.6834155E-0002,-1.2621794E-0002,-
8.4126983E-0003,-4.2057969E-0003
.float -5.8056342E-0021, 4.2057969E-0003, 8.4126983E-0003,
1.2621794E-0002, 1.6834155E-0002
.float 2.1050807E-0002, 2.5272740E-0002, 2.9500874E-0002,
3.3736061E-0002, 3.7979074E-0002
.float 4.2230591E-0002, 4.6491187E-0002, 5.0761323E-0002,
5.5041339E-0002, 5.9331447E-0002
.float 6.3631713E-0002, 6.7942061E-0002, 7.2262250E-0002,
7.6591901E-0002, 8.0930442E-0002
.float 8.5277148E-0002, 8.9631110E-0002, 9.3991242E-0002,
9.8356277E-0002, 1.0272476E-0001
.float 1.0709505E-0001, 1.1146532E-0001, 1.1583357E-0001,
1.2019760E-0001, 1.2455504E-0001
.float 1.2890331E-0001, 1.3323973E-0001, 1.3756135E-0001,
1.4186513E-0001, 1.4614783E-0001

```

.float 1.5040608E-0001, 1.5463634E-0001, 1.5883496E-0001,
1.6299814E-0001, 1.6712198E-0001
.float 1.7120244E-0001, 1.7523539E-0001, 1.7921667E-0001,
1.8314195E-0001, 1.8700688E-0001
.float 1.9080707E-0001, 1.9453810E-0001, 1.9819546E-0001,
2.0177469E-0001, 2.0527130E-0001
.float 2.0868082E-0001, 2.1199881E-0001, 2.1522085E-0001,
2.1834262E-0001, 2.2135980E-0001
.float 2.2426821E-0001, 2.2706375E-0001, 2.2974238E-0001,
2.3230027E-0001, 2.3473363E-0001
.float 2.3703885E-0001, 2.3921251E-0001, 2.4125132E-0001,
2.4315214E-0001, 2.4491209E-0001
.float 2.4652843E-0001, 2.4799864E-0001, 2.4932042E-0001,
2.5049168E-0001, 2.5151059E-0001
.float 2.5237551E-0001, 2.5308508E-0001, 2.5363812E-0001,
2.5403380E-0001, 2.5427148E-0001
.float 2.5435072E-0001, 2.5427148E-0001, 2.5403380E-0001,
2.5363812E-0001, 2.5308508E-0001
.float 2.5237551E-0001, 2.5151059E-0001, 2.5049168E-0001,
2.4932042E-0001, 2.4799864E-0001
.float 2.4652843E-0001, 2.4491209E-0001, 2.4315214E-0001,
2.4125132E-0001, 2.3921251E-0001
.float 2.3703885E-0001, 2.3473363E-0001, 2.3230027E-0001,
2.2974238E-0001, 2.2706375E-0001
.float 2.2426821E-0001, 2.2135980E-0001, 2.1834262E-0001,
2.1522085E-0001, 2.1199881E-0001
.float 2.0868082E-0001

```
.title "IITAB"  
; Double integral of input voltage look-up table. Contains a floating  
point look-up table of the integral of the input voltage from 30ø to  
210ø with 180 steps per cycle.
```

```
.globl _iitab_0  
_iitab_0:  
.float 2.0905461E-0005, 1.1467186E+0000, 4.6294231E+0000,  
1.0507225E+0001, 1.8832653E+0001  
.float 2.9651525E+0001, 4.3002823E+0001, 5.8918591E+0001,  
7.7423843E+0001, 9.8536491E+0001  
.float 1.2226731E+0002, 1.4861989E+0002, 1.7759064E+0002,  
2.0916879E+0002, 2.4333641E+0002  
.float 2.8006851E+0002, 3.1933304E+0002, 3.6109100E+0002,  
4.0529660E+0002, 4.5189731E+0002  
.float 5.0083401E+0002, 5.5206409E+0002, 6.0563019E+0002,  
6.6159137E+0002, 7.2000018E+0002  
.float 7.8090247E+0002, 8.4433716E+0002, 9.1033630E+0002,  
9.7892493E+0002, 1.0501210E+0003  
.float 1.1239353E+0003, 1.2003712E+0003, 1.2794254E+0003,  
1.3610869E+0003, 1.4453379E+0003  
.float 1.5321534E+0003, 1.6215013E+0003, 1.7133428E+0003,  
1.8076317E+0003, 1.9043157E+0003  
.float 2.0033359E+0003, 2.1046035E+0003, 2.2079331E+0003,  
2.3131094E+0003, 2.4199128E+0003  
.float 2.5281199E+0003, 2.6375022E+0003, 2.7478284E+0003,  
2.8588652E+0003, 2.9703767E+0003  
.float 3.0821257E+0003, 3.1938748E+0003, 3.3053860E+0003,  
3.4164229E+0003, 3.5267490E+0003  
.float 3.6361316E+0003, 3.7443384E+0003, 3.8511423E+0003,  
3.9563184E+0003, 4.0596477E+0003  
.float 4.1609155E+0003
```

```
.title "ITAB"  
; Integral of input voltage look-up table. Contains a floating point  
look-up table of the integral of the input voltage from 30ø to 210ø  
with SAMP steps per cycle.
```

```
.globl _itab_0  
_itab_0:  
.float 1.0251468E-0006, 2.3043132E+0000, 4.6709466E+0000,  
7.0934153E+0000, 9.5650787E+0000  
.float 1.2079163E+0001, 1.4628776E+0001, 1.7206930E+0001,  
1.9806559E+0001, 2.2420538E+0001  
.float 2.5041700E+0001, 2.7662863E+0001, 3.0276840E+0001,  
3.2876469E+0001, 3.5454624E+0001  
.float 3.8004238E+0001, 4.0518322E+0001, 4.2989983E+0001,  
4.5412453E+0001, 4.7779087E+0001  
.float 5.0083397E+0001, 5.2387711E+0001, 5.4754345E+0001,  
5.7176811E+0001, 5.9648476E+0001  
.float 6.2162560E+0001, 6.4712173E+0001, 6.7290329E+0001,  
6.9889954E+0001, 7.2503937E+0001  
.float 7.5125099E+0001, 7.7746262E+0001, 8.0360237E+0001,  
8.2959869E+0001, 8.5538017E+0001  
.float 8.8087631E+0001, 9.0601715E+0001, 9.3073380E+0001,  
9.5495850E+0001, 9.7862480E+0001  
.float 1.0016679E+0002, 1.0233383E+0002, 1.0428902E+0002,  
1.0602699E+0002, 1.0754299E+0002  
.float 1.0883286E+0002, 1.0989306E+0002, 1.1072069E+0002,  
1.1131348E+0002, 1.1166980E+0002  
.float 1.1178869E+0002, 1.1166980E+0002, 1.1131348E+0002,  
1.1072069E+0002, 1.0989305E+0002  
.float 1.0883286E+0002, 1.0754298E+0002, 1.0602699E+0002,  
1.0428901E+0002, 1.0233382E+0002  
.float 1.0016679E+0002
```

```

.title "MAINSTAB"
; Mains voltage look-up table. A floating point mains phase voltage
look-up table with SAMP steps per cycle. The range is -SAMP/12 to
+SAMP/12.
; The table is used in the CROSS routine.

.globl _mainsv_0
_mainsv_0 .float 2.5455850E+0002, 2.3113428E+0002, 2.0707654E+0002,
1.8245122E+0002, 1.5732581E+0002
.float 1.3176918E+0002, 1.0585138E+0002, 7.9643448E+0001,
5.3217216E+0001, 2.6645124E+0001
.float -0.0000000E+0000, -2.6645124E+0001, -5.3217216E+0001, -
7.9643448E+0001, -1.0585138E+0002
.float -1.3176918E+0002, -1.5732581E+0002, -1.8245122E+0002, -
2.0707654E+0002, -2.3113428E+0002
.float -2.5455850E+0002

```

```

.title "OUTVTAB"
; Output voltage look-up table. a floating point output phase voltage
look-up table with 300 steps per cycle. The range is -30 to 180.
; The values are the actual output phase voltage/(2^16*415) at the
frequency where speed_old = 2^16.
; This is a cosine wave with 1/6 third harmonic.
; The values are multiplied by speed and motor_volts to get the actual
output voltage.
; The table is used in the CROSS routine.

```

```

.globl _outv_0
_outv_0 .float 7.4277273E-0006, 7.4456616E-0006, 7.4590744E-0006,
7.4682703E-0006, 7.4735576E-0006
.float 7.4752497E-0006, 7.4736631E-0006, 7.4691152E-0006,
7.4619238E-0006, 7.4524046E-0006
.float 7.4408717E-0006, 7.4276345E-0006, 7.4129971E-0006,
7.3972565E-0006, 7.3807028E-0006
.float 7.3636156E-0006, 7.3462647E-0006, 7.3289084E-0006,
7.3117917E-0006, 7.2951466E-0006
.float 7.2791904E-0006, 7.2641237E-0006, 7.2501316E-0006,
7.2373819E-0006, 7.2260236E-0006
.float 7.2161879E-0006, 7.2079865E-0006, 7.2015118E-0006,
7.1968348E-0006, 7.1940080E-0006
.float 7.1930626E-0006, 7.1940080E-0006, 7.1968348E-0006,
7.2015118E-0006, 7.2079865E-0006
.float 7.2161879E-0006, 7.2260236E-0006, 7.2373819E-0006,
7.2501316E-0006, 7.2641237E-0006
.float 7.2791904E-0006, 7.2951466E-0006, 7.3117917E-0006,
7.3289084E-0006, 7.3462647E-0006
.float 7.3636156E-0006, 7.3807028E-0006, 7.3972565E-0006,
7.4129971E-0006, 7.4276345E-0006
.float 7.4408717E-0006, 7.4524046E-0006, 7.4619238E-0006,
7.4691152E-0006, 7.4736631E-0006
.float 7.4752497E-0006, 7.4735576E-0006, 7.4682703E-0006,
7.4590744E-0006, 7.4456616E-0006
.float 7.4277273E-0006, 7.4049758E-0006, 7.3771184E-0006,
7.3438769E-0006, 7.3049823E-0006
.float 7.2601797E-0006, 7.2092262E-0006, 7.1518939E-0006,
7.0879701E-0006, 7.0172582E-0006
.float 6.9395801E-0006, 6.8547752E-0006, 6.7627025E-0006,
6.6632410E-0006, 6.5562913E-0006
.float 6.4417732E-0006, 6.3196298E-0006, 6.1898268E-0006,
6.0523512E-0006, 5.9072145E-0006
.float 5.7544498E-0006, 5.5941150E-0006, 5.4262896E-0006,
5.2510773E-0006, 5.0686044E-0006
.float 4.8790203E-0006, 4.6824966E-0006, 4.4792268E-0006,
4.2694264E-0006, 4.0533305E-0006
.float 3.8311964E-0006, 3.6032984E-0006, 3.3699318E-0006,
3.1314082E-0006, 2.8880565E-0006
.float 2.6402213E-0006, 2.3882617E-0006, 2.1325504E-0006,
1.8734727E-0006, 1.6114243E-0006
.float 1.3468115E-0006, 1.0800484E-0006, 8.1155645E-0007,
5.4176297E-0007, 2.7109940E-0007
.float 4.6792404E-0025,-2.7109940E-0007,-5.4176297E-0007,-
8.1155645E-0007,-1.0800484E-0006
.float -1.3468115E-0006,-1.6114243E-0006,-1.8734727E-0006,-
2.1325504E-0006,-2.3882617E-0006
.float -2.6402213E-0006,-2.8880565E-0006,-3.1314082E-0006,-
3.3699318E-0006,-3.6032984E-0006

```

.float -3.8311964E-0006,-4.0533305E-0006,-4.2694264E-0006,-
 4.4792268E-0006,-4.6824966E-0006
 .float -4.8790203E-0006,-5.0686044E-0006,-5.2510773E-0006,-
 5.4262896E-0006,-5.5941150E-0006
 .float -5.7544498E-0006,-5.9072145E-0006,-6.0523512E-0006,-
 6.1898268E-0006,-6.3196298E-0006
 .float -6.4417732E-0006,-6.5562913E-0006,-6.6632410E-0006,-
 6.7627025E-0006,-6.8547752E-0006
 .float -6.9395801E-0006,-7.0172582E-0006,-7.0879701E-0006,-
 7.1518939E-0006,-7.2092262E-0006
 .float -7.2601797E-0006,-7.3049823E-0006,-7.3438769E-0006,-
 7.3771184E-0006,-7.4049758E-0006
 .float -7.4277273E-0006,-7.4456616E-0006,-7.4590744E-0006,-
 7.4682703E-0006,-7.4735576E-0006
 .float -7.4752497E-0006,-7.4736631E-0006,-7.4691152E-0006,-
 7.4619238E-0006,-7.4524046E-0006
 .float -7.4408717E-0006,-7.4276345E-0006,-7.4129971E-0006,-
 7.3972565E-0006,-7.3807028E-0006
 .float -7.3636156E-0006,-7.3462647E-0006,-7.3289084E-0006,-
 7.3117917E-0006,-7.2951466E-0006
 .float -7.2791904E-0006,-7.2641237E-0006,-7.2501316E-0006,-
 7.2373819E-0006,-7.2260236E-0006
 .float -7.2161879E-0006,-7.2079865E-0006,-7.2015118E-0006,-
 7.1968348E-0006,-7.1940080E-0006
 .float -7.1930626E-0006,-7.1940080E-0006,-7.1968348E-0006,-
 7.2015118E-0006,-7.2079865E-0006
 .float -7.2161879E-0006,-7.2260236E-0006,-7.2373819E-0006,-
 7.2501316E-0006,-7.2641237E-0006
 .float -7.2791904E-0006,-7.2951466E-0006,-7.3117917E-0006,-
 7.3289084E-0006,-7.3462647E-0006
 .float -7.3636156E-0006,-7.3807028E-0006,-7.3972565E-0006,-
 7.4129971E-0006,-7.4276345E-0006
 .float -7.4408717E-0006,-7.4524046E-0006,-7.4619238E-0006,-
 7.4691152E-0006,-7.4736631E-0006
 .float -7.4752497E-0006,-7.4735576E-0006,-7.4682703E-0006,-
 7.4590744E-0006,-7.4456616E-0006
 .float -7.4277273E-0006,-7.4049758E-0006,-7.3771184E-0006,-
 7.3438769E-0006,-7.3049823E-0006
 .float -7.2601797E-0006,-7.2092262E-0006,-7.1518939E-0006,-
 7.0879701E-0006,-7.0172582E-0006
 .float -6.9395801E-0006,-6.8547752E-0006,-6.7627025E-0006,-
 6.6632410E-0006,-6.5562913E-0006
 .float -6.4417732E-0006,-6.3196298E-0006,-6.1898268E-0006,-
 6.0523512E-0006,-5.9072145E-0006
 .float -5.7544498E-0006,-5.5941150E-0006,-5.4262896E-0006,-
 5.2510773E-0006,-5.0686044E-0006
 .float -4.8790203E-0006,-4.6824966E-0006,-4.4792268E-0006,-
 4.2694264E-0006,-4.0533305E-0006
 .float -3.8311964E-0006,-3.6032984E-0006,-3.3699318E-0006,-
 3.1314082E-0006,-2.8880565E-0006
 .float -2.6402213E-0006,-2.3882617E-0006,-2.1325504E-0006,-
 1.8734727E-0006,-1.6114243E-0006
 .float -1.3468115E-0006,-1.0800484E-0006,-8.1155645E-0007,-
 5.4176297E-0007,-2.7109940E-0007
 .float 1.5597468E-0025, 2.7109940E-0007, 5.4176297E-0007,
 8.1155645E-0007, 1.0800484E-0006
 .float 1.3468115E-0006, 1.6114243E-0006, 1.8734727E-0006,
 2.1325504E-0006, 2.3882617E-0006
 .float 2.6402213E-0006, 2.8880565E-0006, 3.1314082E-0006,
 3.3699318E-0006, 3.6032984E-0006

.float 3.8311964E-0006, 4.0533305E-0006, 4.2694264E-0006,
4.4792268E-0006, 4.6824966E-0006
.float 4.8790203E-0006, 5.0686044E-0006, 5.2510773E-0006,
5.4262896E-0006, 5.5941150E-0006
.float 5.7544498E-0006, 5.9072145E-0006, 6.0523512E-0006,
6.1898268E-0006, 6.3196298E-0006
.float 6.4417732E-0006, 6.5562913E-0006, 6.6632410E-0006,
6.7627025E-0006, 6.8547752E-0006
.float 6.9395801E-0006, 7.0172582E-0006, 7.0879701E-0006,
7.1518939E-0006, 7.2092262E-0006
.float 7.2601797E-0006, 7.3049823E-0006, 7.3438769E-0006,
7.3771184E-0006, 7.4049758E-0006
.float 7.4277273E-0006, 7.4456616E-0006, 7.4590744E-0006,
7.4682703E-0006, 7.4735576E-0006
.float 7.4752497E-0006, 7.4736631E-0006, 7.4691152E-0006,
7.4619238E-0006, 7.4524046E-0006
.float 7.4408717E-0006, 7.4276345E-0006, 7.4129971E-0006,
7.3972565E-0006, 7.3807028E-0006
.float 7.3636156E-0006, 7.3462647E-0006, 7.3289084E-0006,
7.3117917E-0006, 7.2951466E-0006
.float 7.2791904E-0006, 7.2641237E-0006, 7.2501316E-0006,
7.2373819E-0006, 7.2260236E-0006
.float 7.2161879E-0006, 7.2079865E-0006, 7.2015118E-0006,
7.1968348E-0006, 7.1940080E-0006
.float 7.1930626E-0006

```
.title "SINTAB"
; Sine look-up table. Contains a floating point sine look-up table of
1.5 cycles length with 300 steps per cycle.
```

```
.globl _sine_0
_sine_0 .float -8.6602539E-0001,-8.7630665E-0001,-8.8620359E-0001,-
8.9571178E-0001,-9.0482706E-0001
.float -9.1354543E-0001,-9.2186314E-0001,-9.2977649E-0001,-
9.3728197E-0001,-9.4437635E-0001
.float -9.5105654E-0001,-9.5731950E-0001,-9.6316254E-0001,-
9.6858317E-0001,-9.7357893E-0001
.float -9.7814763E-0001,-9.8228723E-0001,-9.8599601E-0001,-
9.8927236E-0001,-9.9211472E-0001
.float -9.9452192E-0001,-9.9649286E-0001,-9.9802673E-0001,-
9.9912286E-0001,-9.9978065E-0001
.float -1.0000000E+0000,-9.9978065E-0001,-9.9912286E-0001,-
9.9802673E-0001,-9.9649286E-0001
.float -9.9452192E-0001,-9.9211472E-0001,-9.8927236E-0001,-
9.8599601E-0001,-9.8228723E-0001
.float -9.7814763E-0001,-9.7357893E-0001,-9.6858317E-0001,-
9.6316254E-0001,-9.5731950E-0001
.float -9.5105654E-0001,-9.4437635E-0001,-9.3728197E-0001,-
9.2977649E-0001,-9.2186314E-0001
.float -9.1354543E-0001,-9.0482706E-0001,-8.9571178E-0001,-
8.8620359E-0001,-8.7630665E-0001
.float -8.6602539E-0001,-8.5536426E-0001,-8.4432793E-0001,-
8.3292127E-0001,-8.2114923E-0001
.float -8.0901700E-0001,-7.9652989E-0001,-7.8369343E-0001,-
7.7051324E-0001,-7.5699508E-0001
.float -7.4314481E-0001,-7.2896862E-0001,-7.1447265E-0001,-
6.9966334E-0001,-6.8454713E-0001
.float -6.6913062E-0001,-6.5342063E-0001,-6.3742399E-0001,-
6.2114775E-0001,-6.0459912E-0001
.float -5.8778524E-0001,-5.7071358E-0001,-5.5339158E-0001,-
5.3582680E-0001,-5.1802701E-0001
.float -5.0000000E-0001,-4.8175368E-0001,-4.6329603E-0001,-
4.4463518E-0001,-4.2577928E-0001
.float -4.0673664E-0001,-3.8751557E-0001,-3.6812454E-0001,-
3.4857205E-0001,-3.2886666E-0001
.float -3.0901700E-0001,-2.8903180E-0001,-2.6891983E-0001,-
2.4868989E-0001,-2.2835086E-0001
.float -2.0791169E-0001,-1.8738131E-0001,-1.6676874E-0001,-
1.4608303E-0001,-1.2533323E-0001
.float -1.0452846E-0001,-8.3677843E-0002,-6.2790520E-0002,-
4.1875653E-0002,-2.0942420E-0002
.float 0.0000000E+0000, 2.0942420E-0002, 4.1875653E-0002,
6.2790520E-0002, 8.3677843E-0002
.float 1.0452846E-0001, 1.2533323E-0001, 1.4608303E-0001,
1.6676874E-0001, 1.8738131E-0001
.float 2.0791169E-0001, 2.2835086E-0001, 2.4868989E-0001,
2.6891983E-0001, 2.8903180E-0001
.float 3.0901700E-0001, 3.2886666E-0001, 3.4857205E-0001,
3.6812454E-0001, 3.8751557E-0001
.float 4.0673664E-0001, 4.2577928E-0001, 4.4463518E-0001,
4.6329603E-0001, 4.8175368E-0001
.float 5.0000000E-0001, 5.1802701E-0001, 5.3582680E-0001,
5.5339158E-0001, 5.7071358E-0001
.float 5.8778524E-0001, 6.0459912E-0001, 6.2114775E-0001,
6.3742399E-0001, 6.5342063E-0001
```

.float 6.6913062E-0001, 6.8454713E-0001, 6.9966334E-0001,
 7.1447265E-0001, 7.2896862E-0001
 .float 7.4314481E-0001, 7.5699508E-0001, 7.7051324E-0001,
 7.8369343E-0001, 7.9652989E-0001
 .float 8.0901700E-0001, 8.2114923E-0001, 8.3292127E-0001,
 8.4432793E-0001, 8.5536426E-0001
 .float 8.6602539E-0001, 8.7630665E-0001, 8.8620359E-0001,
 8.9571178E-0001, 9.0482706E-0001
 .float 9.1354543E-0001, 9.2186314E-0001, 9.2977649E-0001,
 9.3728197E-0001, 9.4437635E-0001
 .float 9.5105654E-0001, 9.5731950E-0001, 9.6316254E-0001,
 9.6858317E-0001, 9.7357893E-0001
 .float 9.7814763E-0001, 9.8228723E-0001, 9.8599601E-0001,
 9.8927236E-0001, 9.9211472E-0001
 .float 9.9452192E-0001, 9.9649286E-0001, 9.9802673E-0001,
 9.9912286E-0001, 9.9978065E-0001
 .float 1.0000000E+0000, 9.9978065E-0001, 9.9912286E-0001,
 9.9802673E-0001, 9.9649286E-0001
 .float 9.9452192E-0001, 9.9211472E-0001, 9.8927236E-0001,
 9.8599601E-0001, 9.8228723E-0001
 .float 9.7814763E-0001, 9.7357893E-0001, 9.6858317E-0001,
 9.6316254E-0001, 9.5731950E-0001
 .float 9.5105654E-0001, 9.4437635E-0001, 9.3728197E-0001,
 9.2977649E-0001, 9.2186314E-0001
 .float 9.1354543E-0001, 9.0482706E-0001, 8.9571178E-0001,
 8.8620359E-0001, 8.7630665E-0001
 .float 8.6602539E-0001, 8.5536426E-0001, 8.4432793E-0001,
 8.3292127E-0001, 8.2114923E-0001
 .float 8.0901700E-0001, 7.9652989E-0001, 7.8369343E-0001,
 7.7051324E-0001, 7.5699508E-0001
 .float 7.4314481E-0001, 7.2896862E-0001, 7.1447265E-0001,
 6.9966334E-0001, 6.8454713E-0001
 .float 6.6913062E-0001, 6.5342063E-0001, 6.3742399E-0001,
 6.2114775E-0001, 6.0459912E-0001
 .float 5.8778524E-0001, 5.7071358E-0001, 5.5339158E-0001,
 5.3582680E-0001, 5.1802701E-0001
 .float 5.0000000E-0001, 4.8175368E-0001, 4.6329603E-0001,
 4.4463518E-0001, 4.2577928E-0001
 .float 4.0673664E-0001, 3.8751557E-0001, 3.6812454E-0001,
 3.4857205E-0001, 3.2886666E-0001
 .float 3.0901700E-0001, 2.8903180E-0001, 2.6891983E-0001,
 2.4868989E-0001, 2.2835086E-0001
 .float 2.0791169E-0001, 1.8738131E-0001, 1.6676874E-0001,
 1.4608303E-0001, 1.2533323E-0001
 .float 1.0452846E-0001, 8.3677843E-0002, 6.2790520E-0002,
 4.1875653E-0002, 2.0942420E-0002
 .float -0.0000000E+0000, -2.0942420E-0002, -4.1875653E-0002, -
 6.2790520E-0002, -8.3677843E-0002
 .float -1.0452846E-0001, -1.2533323E-0001, -1.4608303E-0001, -
 1.6676874E-0001, -1.8738131E-0001
 .float -2.0791169E-0001, -2.2835086E-0001, -2.4868989E-0001, -
 2.6891983E-0001, -2.8903180E-0001
 .float -3.0901700E-0001, -3.2886666E-0001, -3.4857205E-0001, -
 3.6812454E-0001, -3.8751557E-0001
 .float -4.0673664E-0001, -4.2577928E-0001, -4.4463518E-0001, -
 4.6329603E-0001, -4.8175368E-0001
 .float -5.0000000E-0001, -5.1802701E-0001, -5.3582680E-0001, -
 5.5339158E-0001, -5.7071358E-0001
 .float -5.8778524E-0001, -6.0459912E-0001, -6.2114775E-0001, -
 6.3742399E-0001, -6.5342063E-0001

.float -6.6913062E-0001,-6.8454713E-0001,-6.9966334E-0001,-
 7.1447265E-0001,-7.2896862E-0001
 .float -7.4314481E-0001,-7.5699508E-0001,-7.7051324E-0001,-
 7.8369343E-0001,-7.9652989E-0001
 .float -8.0901700E-0001,-8.2114923E-0001,-8.3292127E-0001,-
 8.4432793E-0001,-8.5536426E-0001
 .float -8.6602539E-0001,-8.7630665E-0001,-8.8620359E-0001,-
 8.9571178E-0001,-9.0482706E-0001
 .float -9.1354543E-0001,-9.2186314E-0001,-9.2977649E-0001,-
 9.3728197E-0001,-9.4437635E-0001
 .float -9.5105654E-0001,-9.5731950E-0001,-9.6316254E-0001,-
 9.6858317E-0001,-9.7357893E-0001
 .float -9.7814763E-0001,-9.8228723E-0001,-9.8599601E-0001,-
 9.8927236E-0001,-9.9211472E-0001
 .float -9.9452192E-0001,-9.9649286E-0001,-9.9802673E-0001,-
 9.9912286E-0001,-9.9978065E-0001
 .float -1.0000000E+0000,-9.9978065E-0001,-9.9912286E-0001,-
 9.9802673E-0001,-9.9649286E-0001
 .float -9.9452192E-0001,-9.9211472E-0001,-9.8927236E-0001,-
 9.8599601E-0001,-9.8228723E-0001
 .float -9.7814763E-0001,-9.7357893E-0001,-9.6858317E-0001,-
 9.6316254E-0001,-9.5731950E-0001
 .float -9.5105654E-0001,-9.4437635E-0001,-9.3728197E-0001,-
 9.2977649E-0001,-9.2186314E-0001
 .float -9.1354543E-0001,-9.0482706E-0001,-8.9571178E-0001,-
 8.8620359E-0001,-8.7630665E-0001
 .float -8.6602539E-0001,-8.5536426E-0001,-8.4432793E-0001,-
 8.3292127E-0001,-8.2114923E-0001
 .float -8.0901700E-0001,-7.9652989E-0001,-7.8369343E-0001,-
 7.7051324E-0001,-7.5699508E-0001
 .float -7.4314481E-0001,-7.2896862E-0001,-7.1447265E-0001,-
 6.9966334E-0001,-6.8454713E-0001
 .float -6.6913062E-0001,-6.5342063E-0001,-6.3742399E-0001,-
 6.2114775E-0001,-6.0459912E-0001
 .float -5.8778524E-0001,-5.7071358E-0001,-5.5339158E-0001,-
 5.3582680E-0001,-5.1802701E-0001
 .float -5.0000000E-0001,-4.8175368E-0001,-4.6329603E-0001,-
 4.4463518E-0001,-4.2577928E-0001
 .float -4.0673664E-0001,-3.8751557E-0001,-3.6812454E-0001,-
 3.4857205E-0001,-3.2886666E-0001
 .float -3.0901700E-0001,-2.8903180E-0001,-2.6891983E-0001,-
 2.4868989E-0001,-2.2835086E-0001
 .float -2.0791169E-0001,-1.8738131E-0001,-1.6676874E-0001,-
 1.4608303E-0001,-1.2533323E-0001
 .float -1.0452846E-0001,-8.3677843E-0002,-6.2790520E-0002,-
 4.1875653E-0002,-2.0942420E-0002
 .float 0.0000000E+0000, 2.0942420E-0002, 4.1875653E-0002,
 6.2790520E-0002, 8.3677843E-0002
 .float 1.0452846E-0001, 1.2533323E-0001, 1.4608303E-0001,
 1.6676874E-0001, 1.8738131E-0001
 .float 2.0791169E-0001, 2.2835086E-0001, 2.4868989E-0001,
 2.6891983E-0001, 2.8903180E-0001
 .float 3.0901700E-0001, 3.2886666E-0001, 3.4857205E-0001,
 3.6812454E-0001, 3.8751557E-0001
 .float 4.0673664E-0001, 4.2577928E-0001, 4.4463518E-0001,
 4.6329603E-0001, 4.8175368E-0001
 .float 5.0000000E-0001, 5.1802701E-0001, 5.3582680E-0001,
 5.5339158E-0001, 5.7071358E-0001
 .float 5.8778524E-0001, 6.0459912E-0001, 6.2114775E-0001,
 6.3742399E-0001, 6.5342063E-0001

```

.float 6.6913062E-0001, 6.8454713E-0001, 6.9966334E-0001,
7.1447265E-0001, 7.2896862E-0001
.float 7.4314481E-0001, 7.5699508E-0001, 7.7051324E-0001,
7.8369343E-0001, 7.9652989E-0001
.float 8.0901700E-0001, 8.2114923E-0001, 8.3292127E-0001,
8.4432793E-0001, 8.5536426E-0001
.float 8.6602539E-0001, 8.7630665E-0001, 8.8620359E-0001,
8.9571178E-0001, 9.0482706E-0001
.float 9.1354543E-0001, 9.2186314E-0001, 9.2977649E-0001,
9.3728197E-0001, 9.4437635E-0001
.float 9.5105654E-0001, 9.5731950E-0001, 9.6316254E-0001,
9.6858317E-0001, 9.7357893E-0001
.float 9.7814763E-0001, 9.8228723E-0001, 9.8599601E-0001,
9.8927236E-0001, 9.9211472E-0001
.float 9.9452192E-0001, 9.9649286E-0001, 9.9802673E-0001,
9.9912286E-0001, 9.9978065E-0001
.float 1.0000000E+0000, 9.9978065E-0001, 9.9912286E-0001,
9.9802673E-0001, 9.9649286E-0001
.float 9.9452192E-0001, 9.9211472E-0001, 9.8927236E-0001,
9.8599601E-0001, 9.8228723E-0001
.float 9.7814763E-0001, 9.7357893E-0001, 9.6858317E-0001,
9.6316254E-0001, 9.5731950E-0001
.float 9.5105654E-0001, 9.4437635E-0001, 9.3728197E-0001,
9.2977649E-0001, 9.2186314E-0001
.float 9.1354543E-0001, 9.0482706E-0001, 8.9571178E-0001,
8.8620359E-0001, 8.7630665E-0001
.float 8.6602539E-0001, 8.5536426E-0001, 8.4432793E-0001,
8.3292127E-0001, 8.2114923E-0001
.float 8.0901700E-0001, 7.9652989E-0001, 7.8369343E-0001,
7.7051324E-0001, 7.5699508E-0001
.float 7.4314481E-0001, 7.2896862E-0001, 7.1447265E-0001,
6.9966334E-0001, 6.8454713E-0001
.float 6.6913062E-0001, 6.5342063E-0001, 6.3742399E-0001,
6.2114775E-0001, 6.0459912E-0001
.float 5.8778524E-0001, 5.7071358E-0001, 5.5339158E-0001,
5.3582680E-0001, 5.1802701E-0001
.float 5.0000000E-0001, 4.8175368E-0001, 4.6329603E-0001,
4.4463518E-0001, 4.2577928E-0001
.float 4.0673664E-0001, 3.8751557E-0001, 3.6812454E-0001,
3.4857205E-0001, 3.2886666E-0001
.float 3.0901700E-0001, 2.8903180E-0001, 2.6891983E-0001,
2.4868989E-0001, 2.2835086E-0001
.float 2.0791169E-0001, 1.8738131E-0001, 1.6676874E-0001,
1.4608303E-0001, 1.2533323E-0001
.float 1.0452846E-0001, 8.3677843E-0002, 6.2790520E-0002,
4.1875653E-0002, 2.0942420E-0002
.float -2.1684043E-0019, -2.0942420E-0002, -4.1875653E-0002, -
6.2790520E-0002, -8.3677843E-0002
.float -1.0452846E-0001, -1.2533323E-0001, -1.4608303E-0001, -
1.6676874E-0001, -1.8738131E-0001
.float -2.0791169E-0001, -2.2835086E-0001, -2.4868989E-0001, -
2.6891983E-0001, -2.8903180E-0001
.float -3.0901700E-0001, -3.2886666E-0001, -3.4857205E-0001, -
3.6812454E-0001, -3.8751557E-0001
.float -4.0673664E-0001, -4.2577928E-0001, -4.4463518E-0001, -
4.6329603E-0001, -4.8175368E-0001
.float -5.0000000E-0001

```