

Real-Time Embedded Systems

CpE-450 Spring 07

Class 9

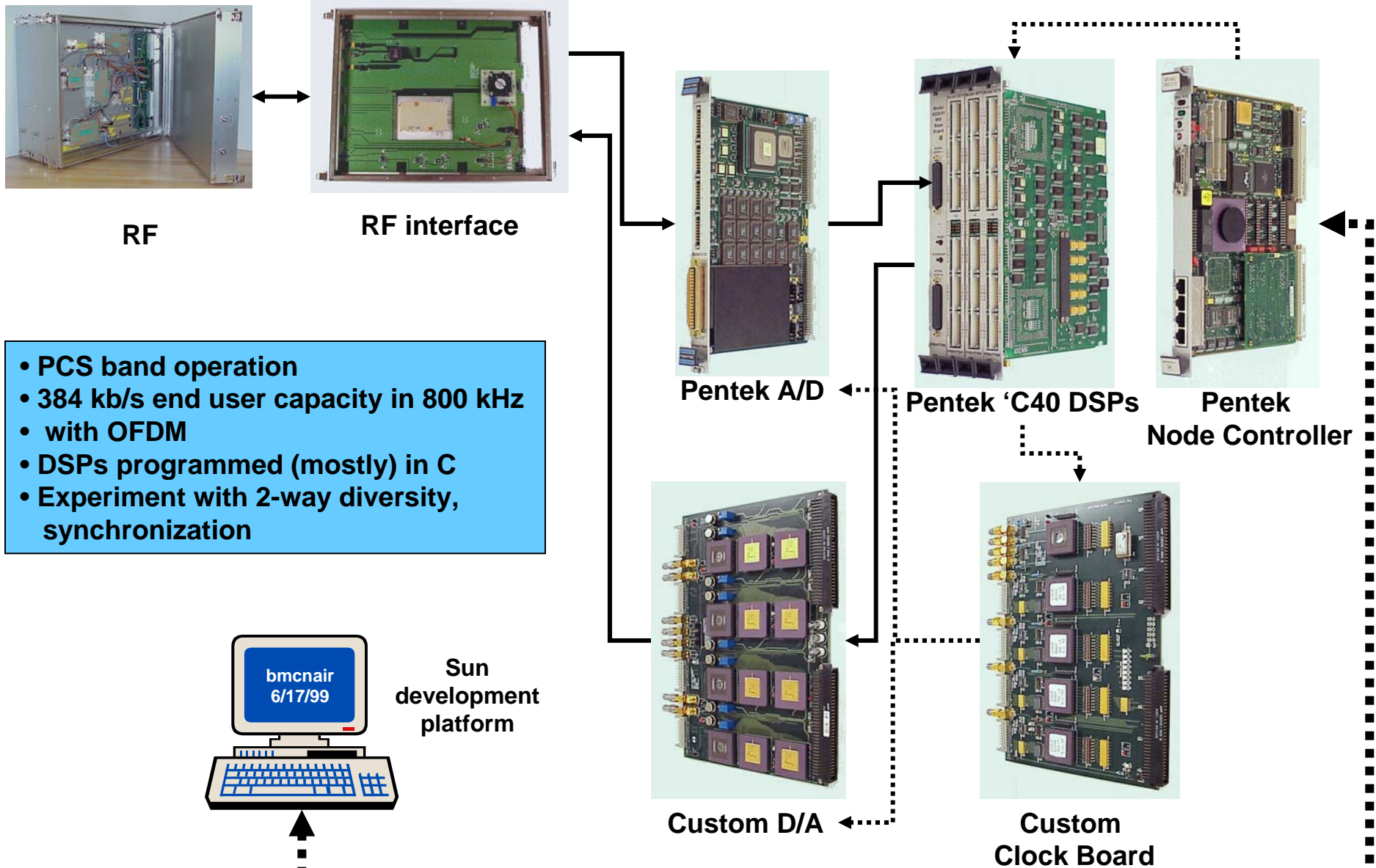
Bruce McNair

bmcnair@stevens.edu

Case Study 2: Real-Time DSP for OFDM Signal Processing

- McNair, B., Cimini, L., Sollenberger, N., "*Performance of an Experimental 384 kb/s 1900 MHz Radio Link In a Wide-Area High-Mobility Environment*," Proc. IEEE Vehicular Technology Conference, VTC00, Boston, MA, October 2000 (<http://www.novidesic.com/pubs/vtc2000-a34283.pdf>).
- McNair, B., Cimini, L., Sollenberger, N., "*Implementation of an Experimental 384 kb/s Radio Link for High-Speed Internet Access*," Proc. IEEE Vehicular Technology Conference, VTC00, Boston, MA, October 2000 (<http://www.novidesic.com/pubs/vtc2000-a41505.pdf>).
- McNair, B., Cimini, L., Sollenberger, N., "*A Robust Timing and Frequency Offset Estimation Scheme for Orthogonal Frequency Division Multiplexing (OFDM) Systems*," Proc. IEEE Vehicular Technology Conference, VTC99, Houston, TX, May 1999(<http://www.novidesic.com/pubs/vtc99-513a.pdf>).

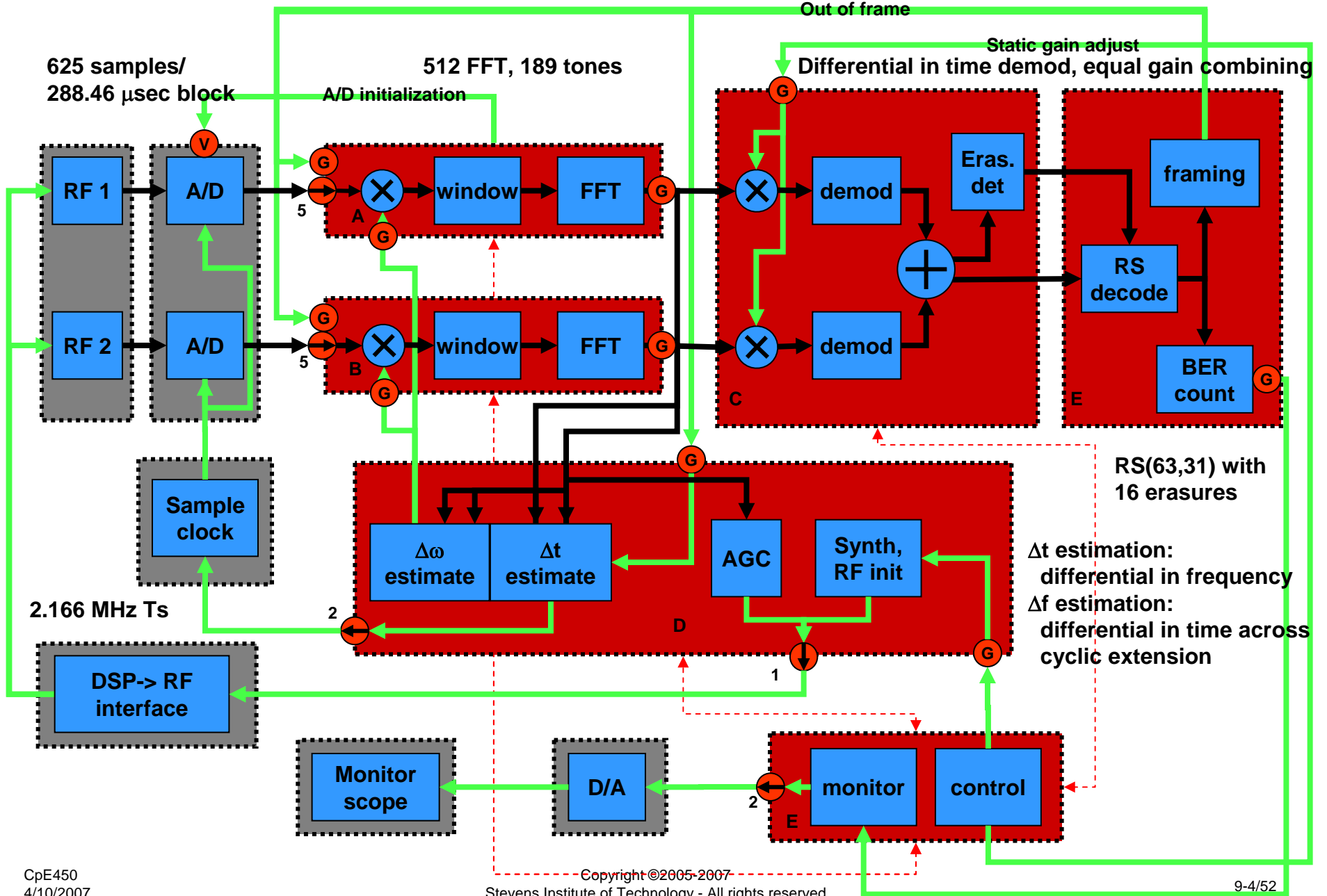
OFDM Prototype

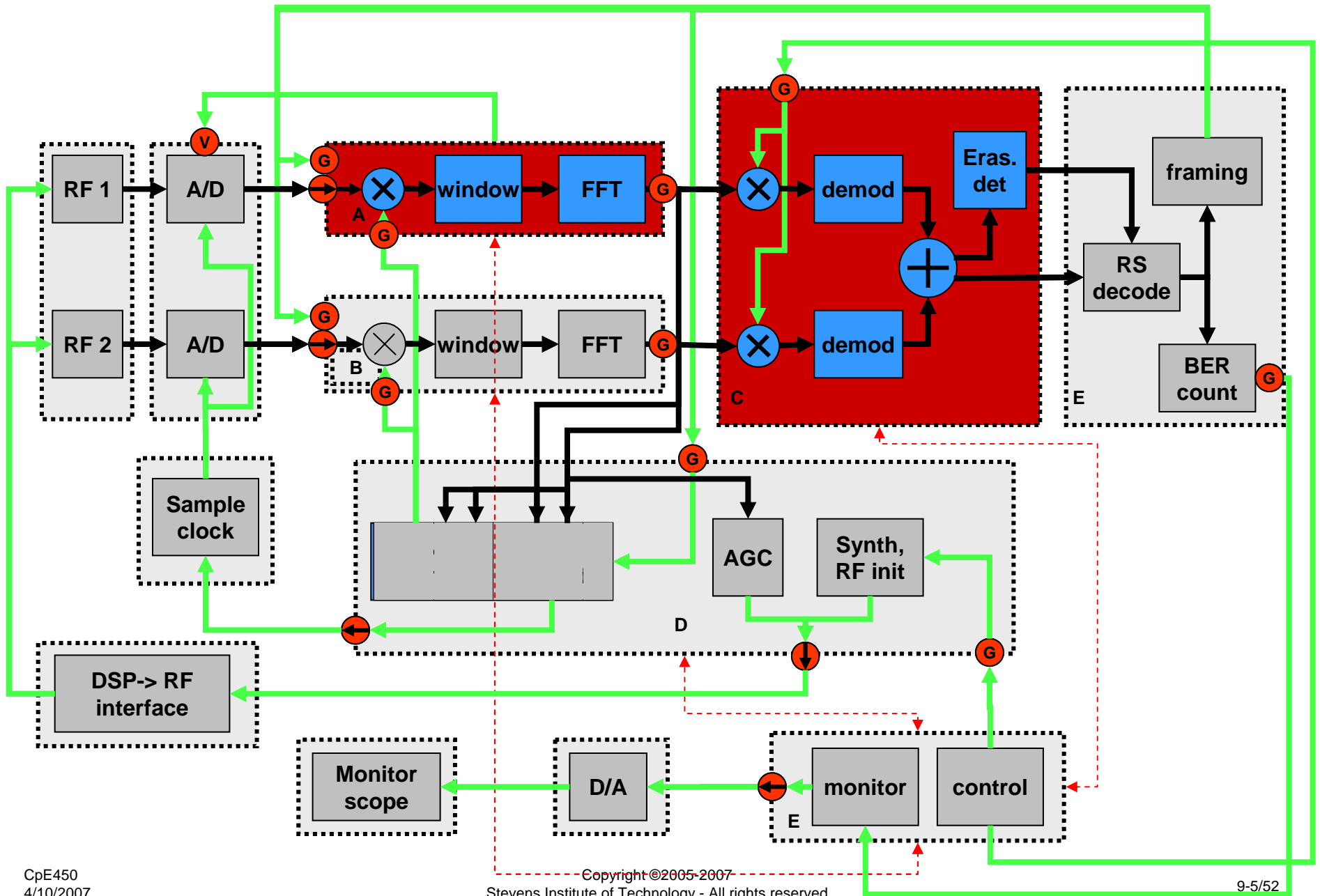


- PCS band operation
- 384 kb/s end user capacity in 800 kHz
- with OFDM
- DSPs programmed (mostly) in C
- Experiment with 2-way diversity, synchronization

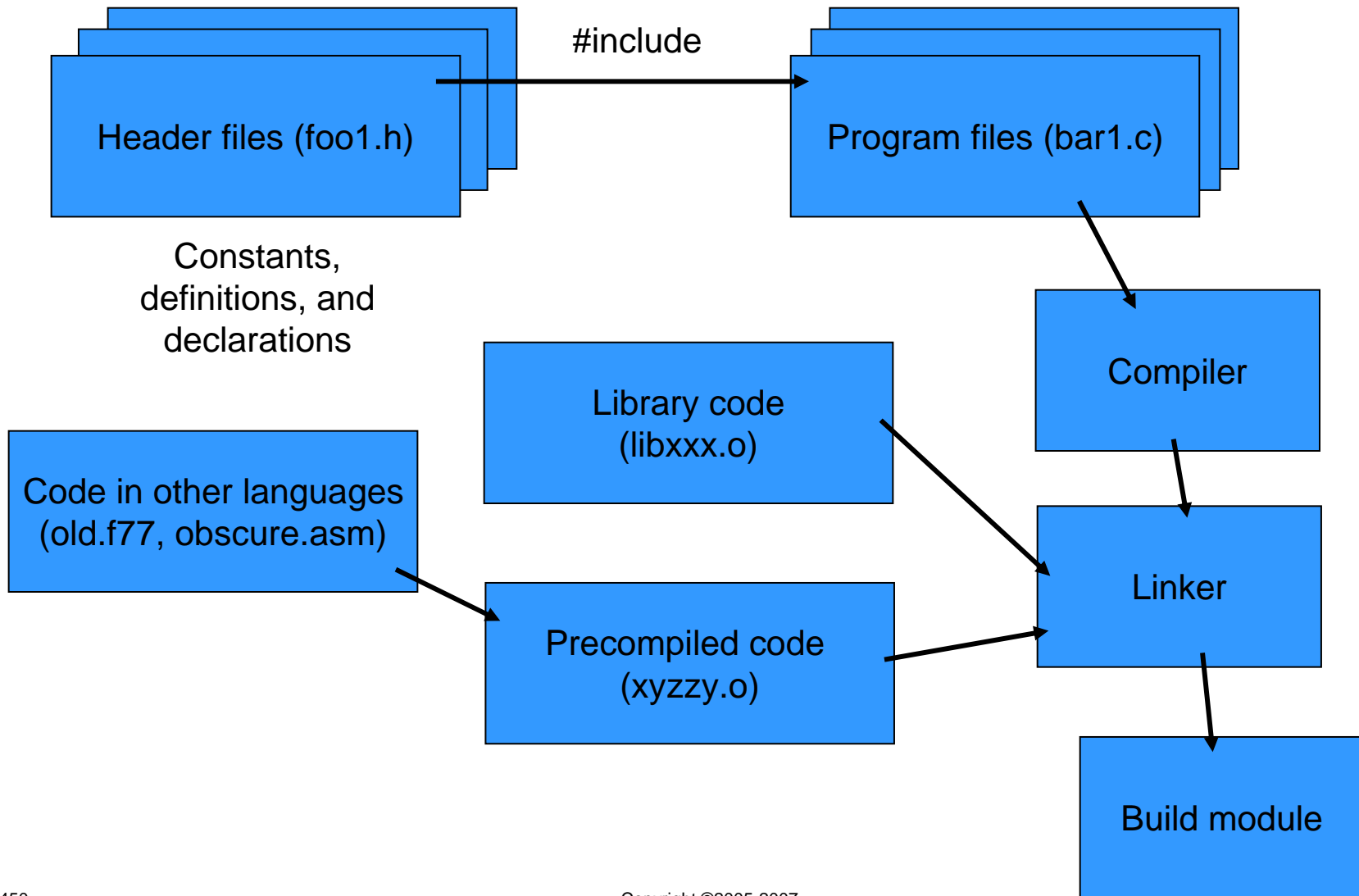
Differential Equal Gain Diversity OFDM Receiver

Hardware Architecture





Building an Executable



Makefiles

```
# this is a Makefile to create the components of the
# acis combined receiver
#
# functions are divided between 5 DSPs
# - there are 1 or 2 DSPs for the FFT frontends
# - 1 DSP for the demodulator/backend
# - 1 DSP for signal control
# - 1 DSP for the user interface for monitoring and controlling.
# added a 6th DSP for decoding
```

```
receiver:      fft demod sigctl monitor decoder
rx:            fft demod sigctl monitor decoder

fft:          fft_frontend
demod:        demod_backend
sigctl:       signal_controls
decoder:      decoder_plus
monitor:      control_monitor

coder:        coder_fn.c
              gcc coder_fn.c -o coder

clean:
rm -f *.o70 *.x70 *.map70 *.s70 *.nfo *.pp *.lst *.l70
rm -f fft_frontend demod_backend signal_controls control_monitor decoder_plus
```

```
fft_frontend:      fft_frontend.c acis_utilities.c acis_interface.h acis_constants.h \
                  compilation_environment.h acis_frame_structure.h asmfft.asm \
                  6109ad.h 6109_modem_parms.h acis_monitor.h \
                  fft192.asm compiler_version_defn.h fft_rx.asm \
                  acis_common.c init_6109.c combined_common.c combined_common.h \
                  6109_modem_parms.h
                  70compile fft_frontend
                  date>>fft_frontend

demod_backend:    demod_backend.c acis_utilities.c acis_interface.h acis_constants.h \
                  compilation_environment.h acis_frame_structure.h asmfft.asm \
                  6109ad.h 6109_modem_parms.h acis_monitor.h \
                  fft192.asm compiler_version_defn.h fft_rx.asm \
                  coder_fn.c fft64asm.asm \
                  acis_common.c init_6109.c combined_common.c combined_common.h
                  70compile demod_backend
                  date>>demod_backend
```

```
receiver:      fft demod sigctl monitor decoder
rx:            fft demod sigctl monitor decoder
fft:          fft_frontend
demod:        demod_backend
```

```
clean:
rm -f *.o70 *.x70 *.map70 *.s70 *.nfo *.pp *.lst *.l70
rm -f fft_frontend demod_backend signal_controls control_monitor decoder_plus
```

Makefiles

```
# this is a Makefile to create the components of the
# acis combined receiver
#
# functions are divided between 5 DSPs
# - there are 1 or 2 DSPs for the FFT frontends
# - 1 DSP for the demodulator/backend
# - 1 DSP for signal control
# - 1 DSP for the user interface for monitoring and controlling.
# added a 6th DSP for decoding
```

```
receiver:          fft_demod sigctl monitor decoder
rx:                fft_demod sigctl monitor decoder

fft:               fft_frontend
demod:             demod_backend
sigctl:           signal_controls
decoder_plus:     decoder_plus
monitor:          control_monitor

coder:             coder_fn.c
                  gcc coder_fn.c -o coder
```

```
clean:
    rm -f *.o70 *.x70 *.map70 *.s70 *.nfo *.pp *.lst *.l70
    rm -f fft_frontend demod_backend signal_controls control_monitor decoder_plus
```

```
fft_frontend:      fft_frontend.c acis_utilities.c acis_interface.h acis_constants.h \
                  compilation_environment.h acis_frame_structure.h asmfft.asm \
                  6109ad.h 6109_modem_params.h acis_monitor.h \
                  fft192.asm compiler_version_defn.h fft_rx.asm \
                  acis_common.c init_6109.c combined_common.c combined_common.h \
                  6109_modem_params.h
                  70compile fft_frontend
                  date>>fft_frontend
```

```
demod_backend:    demod_backend.c acis_utilities.c acis_interface.h acis_constants.h \
                  compilation_environment.h acis_frame_structure.h asmfft.asm \
                  6109ad.h 6109_modem_params.h acis_monitor.h \
                  fft192.asm compiler_version_defn.h fft_rx.asm \
                  coder_fn.c fft64asm.asm \
                  acis_common.c init_6109.c combined_common.c combined_common.h
                  70compile demod_backend
                  date>>demod_backend
```

```
fft_frontend:      fft_frontend.c acis_utilities.c acis_interface.h \
                  acis_constants.h compilation_environment.h acis_frame_structure.h \
                  asmfft.asm 6109ad.h 6109_modem_params.h acis_monitor.h \
                  fft192.asm compiler_version_defn.h fft_rx.asm \
                  acis_common.c init_6109.c combined_common.c combined_common.h \
                  6109_modem_params.h
                  70compile fft_frontend
                  date>>fft_frontend
```

Which Was The Version Of The Code That Was Working???


```
/* this is the header file that defines all interfaces between
 * submodules of the ACIS system . This is based on
 * a header file from the E-IS136 project
 *
 * @(#)acis_interface.h 1.1 - 2/4/98 16:56:51
 *
 * Revisions:
 * 1.1 - baseline code
 *
 */

#define full_int_range 2147483648
#define MSH 0xFFFF0000
#define LSH 0x0000FFFF
#define DA_write(A,B,C) #define channel_B(A) ((##A##&LSH)<<16)&MSH
#define n2 9.313225746e-10 /* normalize to 2.0 */
#define xtrt_chA(A) n2*(int)(##A##&MSH)
#define xtrt_chB(A) n2*(int)((##A##&LSH)<<16)&MSH

#define AD_read_both(A,B,C,D)

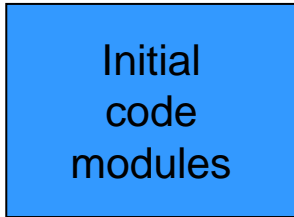
#define OFFSET_6102_INIT 0xDF55
```

Software Version Control



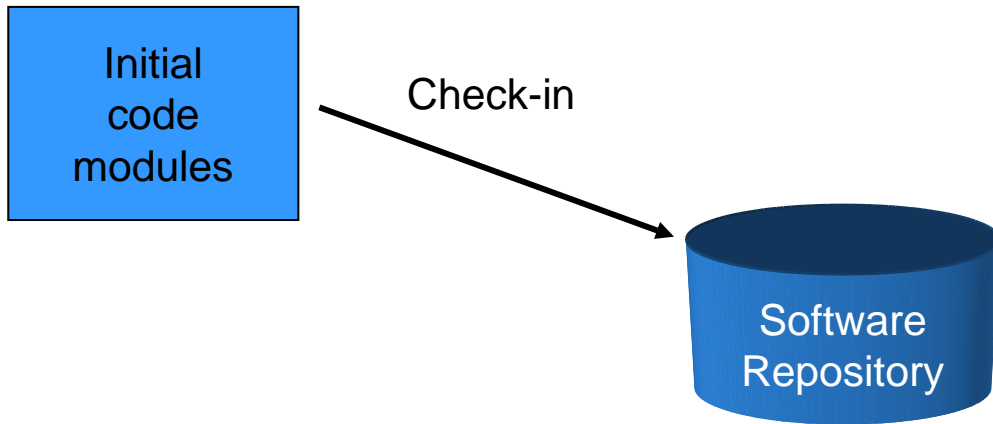
UNIX/linux	Windows
SCCS – Source Code Control System (GNU – CSSC)	
RCS – Revision Control System	
	SourceSafe™
CVS – Concurrent Version System	
Subversion	
...	

Software Version Control

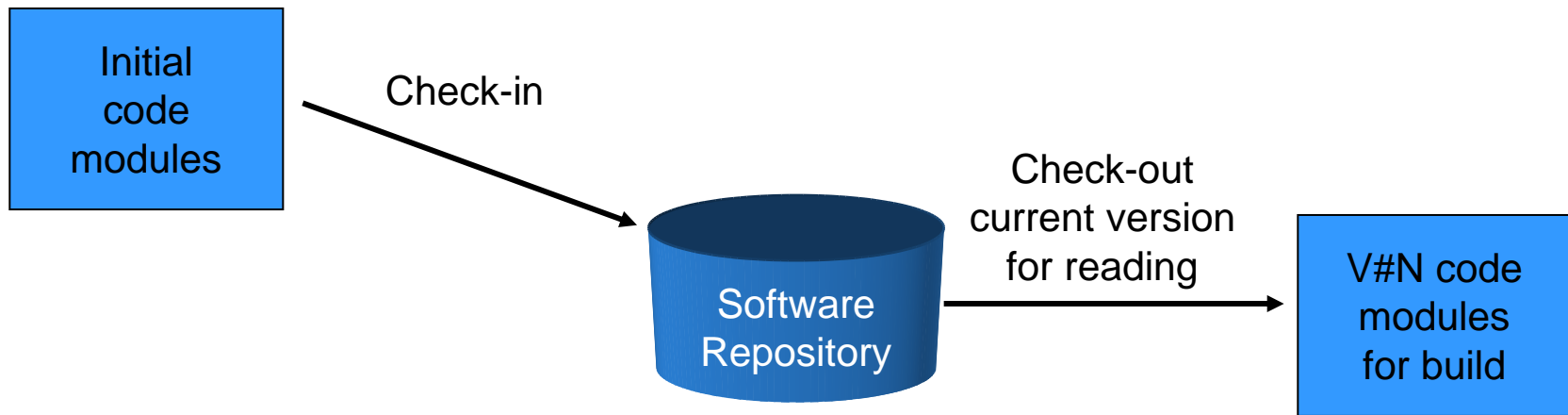


Initial
code
modules

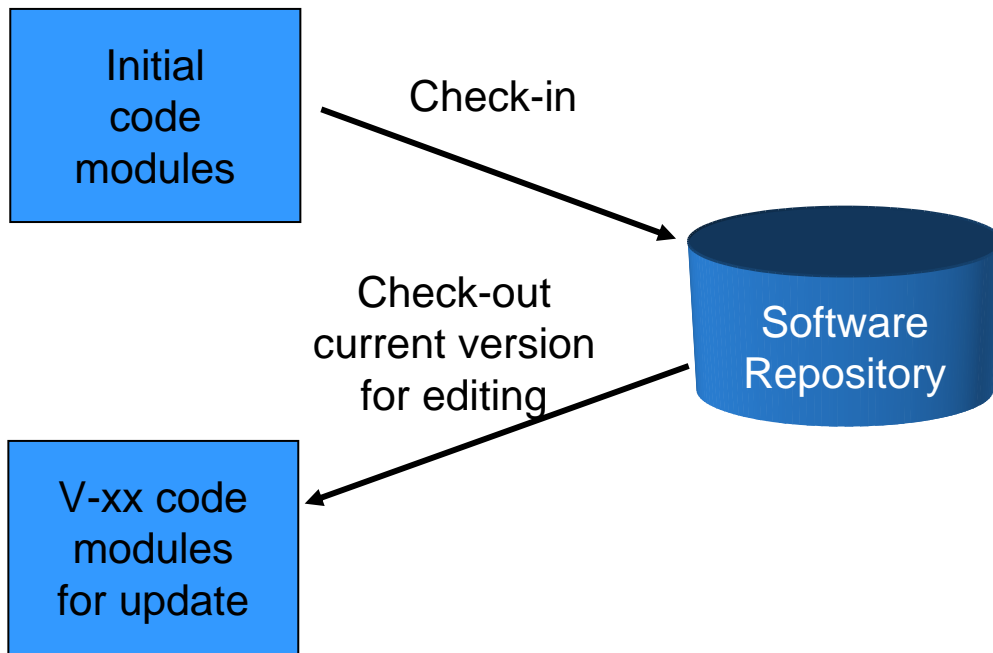
Software Version Control



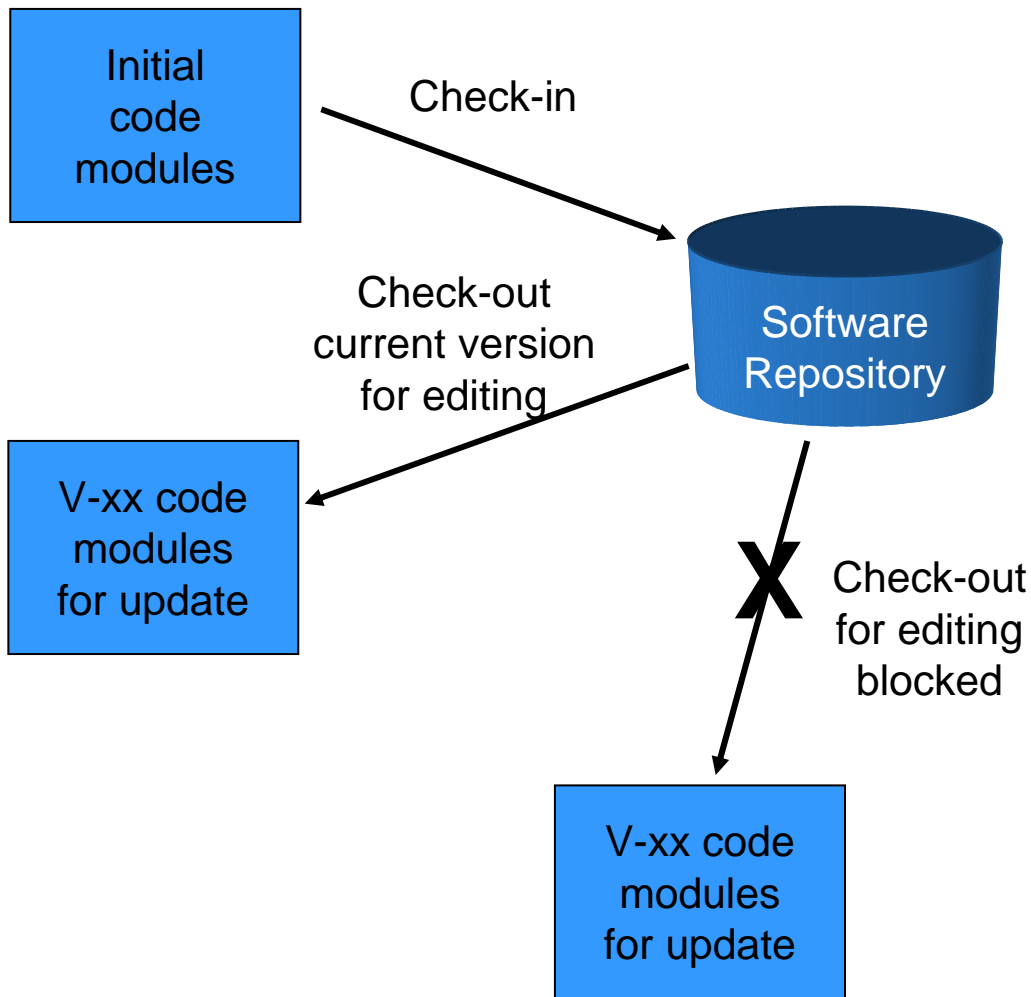
Software Version Control



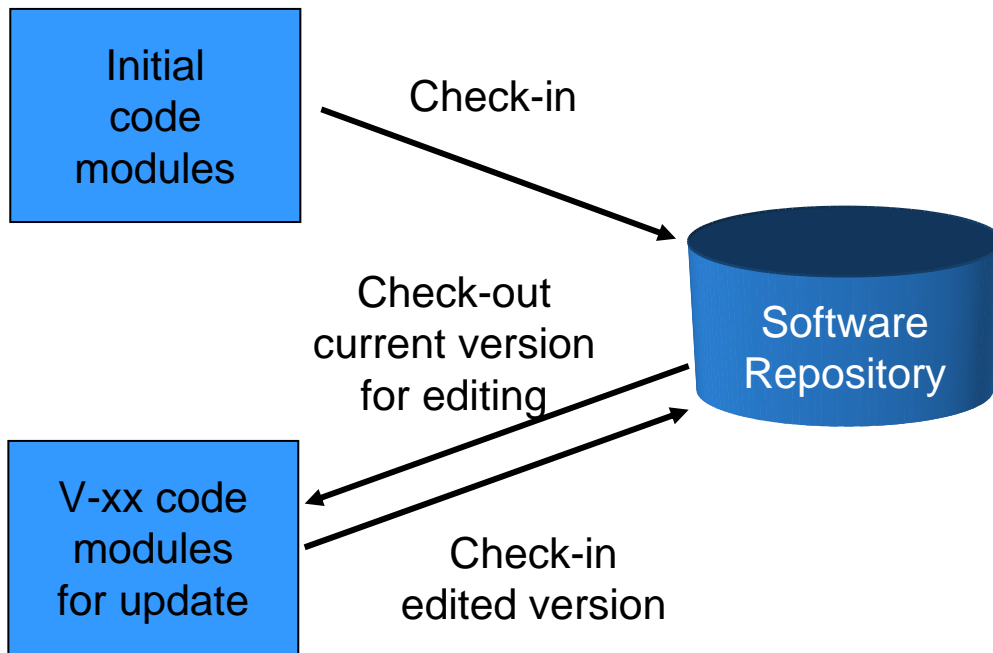
Software Version Control



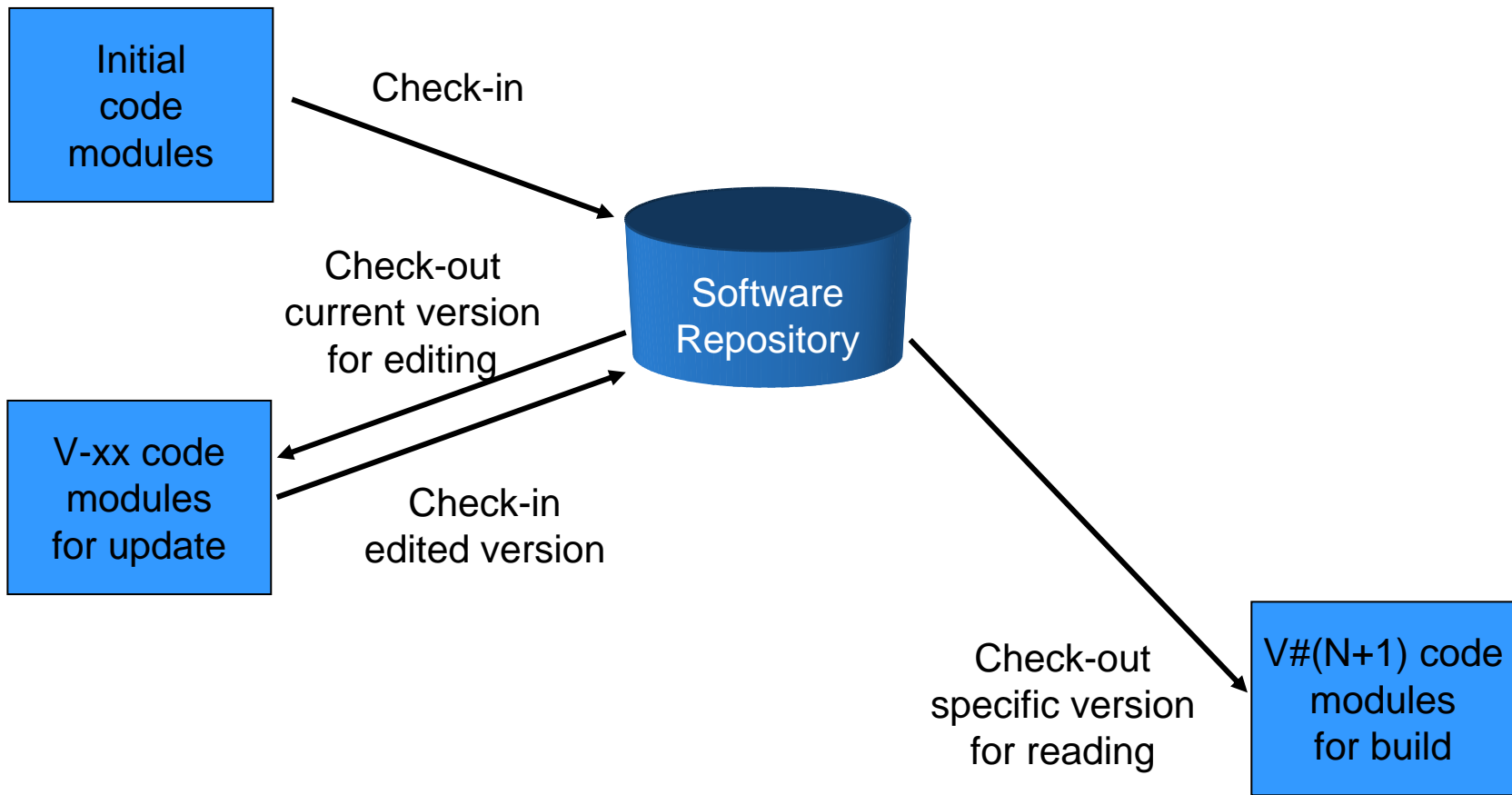
Software Version Control



Software Version Control



Software Version Control



Interrupt Handler

```
void c_int01(void)
{
    int i;
    static int buffer_count;
    static int my_function;
    static int loop_count=0;
    static volatile int time_counter=0;
    static volatile int *start_time;
    static volatile int *event_time;
    static volatile int *stored_count;
    static volatile int tclock_counter=0;

    /* disable DMA interrupt */
    reset_ii(DMA_DONE);
    toggle_tclk1();

    *foreground_block_count += 1;
    interrupts_processed += 1;
```

Interrupt Handler

```
if(ISR_first_time)
{
    buffer_count=0;
    my_function = whats_my_function();

    /* synchronize the two DSPs to the same
count so they blink together */
    loop_count = *(int *)0x80089000;

    /* the particular value is not important, as long
as it is positive */
    if(loop_count < 0)
        loop_count = 0;

    ISR_first_time = FALSE;

    /* start timer on first interrupt */
    #if (TIMER1_FUNCTION == RUN_TIMER)
        time_start(1);
    #endif
}
```

Interrupt Handler

```
/* signal the other processors that a block has ended.
 * This will be used, for instance, to insure that the
 * AGC signal is written before the start of another
 * block
 */
if(my_function == FFT_FRONTEND1)
{
    toggle(BLOCK_BOUNDARY);
}
if(time_counter++ > 999)
{
    *event_time = time_read(1);
    time_counter = 0;
    *stored_count += 1;
}

blocks_read++;
```

Interrupt Handler

```
if(loop_count == POINT9_REAL_OFDM_BLOCKS_PER_SECOND)
{
    toggle_led();
}
if(loop_count++ >= REAL_OFDM_BLOCKS_PER_SECOND)
{
    toggle_led();
    loop_count=0;
}

/* point to the next OFDM block */
interrupt_buffer_pointer += EXTENDED_SAMPLES_PER_BLOCK;
buffer_count += 1;
```

Interrupt Handler

```
if(drop_a_block_for_frame_sync)
{
if(theres_a_second_frontend)
{
if(my_function == FFT_FRONTEND1)
{
snooze(10,usec);
reset_semaphore(FFT2_HEARD_BLOCK_DROP_REQUEST);
signal(FFT1_HEARD_BLOCK_DROP_REQUEST);
snooze(20, usec);
if(test_semaphore(FFT2_HEARD_BLOCK_DROP_REQUEST))
{
signal(PERFORM_BLOCK_DROP);
wait(BLOCK_DROP_ACK);
drop_a_block_for_frame_sync = FALSE;
buffer_count += blocks_to_skip;
interrupt_buffer_pointer += blocks_to_skip*EXTENDED_SAMPLES_PER_BLOCK;
}
}
}
}
```

Interrupt Handler

```
else
{
  reset_semaphore(FFT1_HEARD_BLOCK_DROP_REQUEST);
  snooze(10,usec);
  signal(FFT2_HEARD_BLOCK_DROP_REQUEST);
  snooze(10,usec);
  if(test_semaphore(FFT1_HEARD_BLOCK_DROP_REQUEST))
  {
    wait(PERFORM_BLOCK_DROP);
    signal(BLOCK_DROP_ACK);
    drop_a_block_for_frame_sync = FALSE;
    buffer_count += blocks_to_skip;
    interrupt_buffer_pointer +=
      blocks_to_skip*EXTENDED_SAMPLES_PER_BLOCK;
  }
}
```

Interrupt Handler

```
else
{
  reset_semaphore(FFT1_HEARD_BLOCK_DROP_REQUEST);
  snooze(10,usec);
  signal(FFT2_HEARD_BLOCK_DROP_REQUEST);
  snooze(10,usec);
  if(test_semaphore(FFT1_HEARD_BLOCK_DROP_REQUEST))
  {
    wait(PERFORM_BLOCK_DROP);
    signal(BLOCK_DROP_ACK);
    drop_a_block_for_frame_sync = FALSE;
    buffer_count += blocks_to_skip;
    interrupt_buffer_pointer +=
      blocks_to_skip*EXTENDED_SAMPLES_PER_BLOCK;
  }
}
```

Interrupt Handler

```
else /* only one frontend */
{
    drop_a_block_for_frame_sync = FALSE;
    buffer_count += blocks_to_skip;
    interrupt_buffer_pointer +=
        blocks_to_skip*EXTENDED_SAMPLES_PER_BLOCK;
}
}
```

```
/* if we have hit the end of the current buffer... */
if(buffer_count >= N_DMA_BUFFERS)
{
    /* was buffer_count = 0 */
    /* in case we are several buffers past the end */
    buffer_count -= N_DMA_BUFFERS;

    /* was interrupt_buffer_pointer = start_of_DMA_buffers */
    interrupt_buffer_pointer -=
(N_DMA_BUFFERS)*(EXTENDED_SAMPLES_PER_BLOCK);
}
```

Interrupt Handler

```
switch(my_function)
{
case FFT_FRONTEND1:
  if( ((int) interrupt_buffer_pointer ) < FFT_FRONTEND1_IBP_ADDR)
  {
    printf("interrupt buffer pointer out of range: 0x%8x\n",
           interrupt_buffer_pointer);
    printf("buffer_count = %d\n", buffer_count);
    exit(0);
  }
  break;
case FFT_FRONTEND2:
  if( ((int) interrupt_buffer_pointer ) < FFT_FRONTEND2_IBP_ADDR)
  {
    printf("interrupt buffer pointer out of range: 0x%8x\n",
           interrupt_buffer_pointer);
    printf("buffer_count = %d\n", buffer_count);
    exit(0);
  }
  break;
}
```

Interrupt Handler

·
·
·
·

```
/* save the buffer counts so the background knows where the ISR is/has been */  
new_ISR_buffer_count = buffer_count-1;
```

```
tclock_counter += 1;
```

```
if(tclock_counter > 2)
```

```
{
```

```
toggle_tclk1();
```

```
}
```

```
if(tclock_counter >= 104)
```

```
{
```

```
tclock_counter = 0;
```

```
}
```

```
/* re-enable the DMA interrupts */
```

```
set_iie(DMA_DONE);
```

```
}
```

Interruptible Routine

```
#include <stdlib.h>
#include "compilation_environment.h"
#if DSP_COMPILE
#include <snstdio.h>
#include <math.h>
#include <intpt40.h>
#include <timer40.h>
#include <dma40.h>
#endif
#if UNIX_COMPILE
#include <stdio.h>
#include <math.h>
#endif
#include "acis_constants.h"
#include "compiler_version_defn.h"
#include "acis_frame_structure.h"
#include "acis_monitor.h"
#include "combined_common.h"
#include "6109_modem_parms.h"

/* this is the fft_frontend for the OFDM Phase 1 prototype
 * it is derived from acis_rx.c V1.19 6/8/99
```

Interruptible Routine

```
#include <stdlib.h>
#include "compilation_environment.h"
#if DSP_COMPILE
#include <snstdio.h>
#include <math.h>
#include <intpt40.h>
#include <timer40.h>
#include <dma40.h>
#endif
#if UNIX_COMPILE
#include <stdio.h>
#include <math.h>
#endif
#include "acis_constants.h"
#include "compiler_version_defn.h"
#include "acis_frame_structure.h"
#include "acis_monitor.h"
#include "combined_common.h"
#include "6109_modem_parms.h"

/* this is the fft_frontend for the OFDM Phase 1 prototype
 * it is derived from acis_rx.c V1.19 6/8/99
```

Interruptible Routine

```
#define RUN_TIMER      0
#define DEBUG_OUTPUT  1
#define TIMER1_FUNCTION      DEBUG_OUTPUT

#if DSP_COMPILE
void c_int01(void);          /* input buffer handler */
void start_DMA_from_comm_port(int);
void initialize_DMA_buffer_pointer(int);
int *start_of_DMA_buffers;
#define N_DMA_BUFFERS BLOCKS_PER_FRAME /* 104 blocks = one 30 ms slot */
int *interrupt_buffer_pointer;
#endif

/* prototype function definitions */
void scope_display(int, COMPLEX *, int, int, int, float, float);

void display_run_parameters(void);
void wait_for_next_block(void);
```

Interruptible Routine

```
.  
.br/>void main(void)  
{  
    volatile int *good_block_marker;  
    volatile int *block_resync_request_count;  
    int disp_count=0;  
    volatile float *time_domain_delta_f_estimate;  
.  
.  
#if DSP_COMPILE  
  
    #if DSP_COMPILER==V4dot70  
        install_int_vector((ISR_FnPtr) c_int01, DMA0_FLAG_OFFSET);  
    #endif  
    #if DSP_COMPILER==V4dot60  
        install_int_vector((void *) c_int01, DMA_DONE);  
    #endif  
    reset_iiie(DMA_DONE);  
    DMA_RESET(0);  
#endif  
  
    led_off();
```

Interruptible Routine

```
initialize_interprocessor_parameters(my_function, &delta_f,  
                                   &drop_a_block_for_frame_sync,  
                                   &drop_samples_for_block_sync);  
  
while(*run_stop_control == STOP)  
{  
    snooze(10, usec);  
}  
  
/* do the handshaking between the FFT DSPs */  
front_end_initial_sync();  
toggle_tclk1();  
  
/* enable DMA interrupt */  
set_iie(DMA_DONE);  
toggle_tclk1();  
  
INT_ENABLE();  
toggle_tclk1();  
  
/* start the initial DMA */  
DMA_shorted = FALSE; /* a full size one */  
start_DMA_from_comm_port(EXTENDED_SAMPLES_PER_BLOCK);  
toggle_tclk1();
```

Interruptible Routine

```
while(RUN_MODE)
{
*background_block_count += 1;

if(background_loop_count++ >= REAL_OFDM_BLOCKS_PER_SECOND)
{
toggle_led();
background_loop_count=0;
}

start_value = interrupts_processed;

get_interprocessor_parameters(my_function, &programmed_delta_f,
                             &drop_a_block_for_frame_sync,
                             &drop_samples_for_block_sync);

.
.
.
```

Recycling Interrupts

```
void start_DMA_from_comm_port(int N_samples)
{
/* each time this routine is called, it will initiate
 * another DMA. Generally this will be called from the ISR
 * the first call will be from main() when the while(1) loop begins
 */

static int block=0;

/* set source address to COMM PORT */
*(int *)DMA_SOURCE_ADDRESS = COMM_IN_ADDRESS;

/* set destination address to DMA buffer */
*(int *)DMA_DEST_ADDRESS = (int) interrupt_buffer_pointer;

/* set transfer counter to OFDM block size - #samples to move */
*(int *)DMA_XFR_COUNTER = N_samples;

/* set source index counter to 0 - always from same COMMPORT FIFO */
*(int *)DMA_SOURCE_INDEX = 0;
```

Recycling Interrupts

```
/* set destination index counter to 1 - sequential address writes */
*(int *)DMA_DEST_INDEX = 1;

#define FAVOR_CPU          0x00  /* bits 0-1 */
#define ROTATING_ARBITRATION 0x02
#define FAVOR_DMA          0x03
#define DMA_PRIORITY       ROTATING_ARBITRATION

#define ONGOING_XFR        0x00  /* bits 2-3 */
#define TERMINATE_WITH_TC  0x04  /* and 4-5 for AUX */
#define AUTOINIT_DMA       0x08
#define AUTOINIT_ON_RESTART 0x0C
#define TRANSFER_MODE      TERMINATE_WITH_TC

#define NO_SYNC            0x00  /* bits 6-7 */
#define SOURCE_SYNC        0x40
#define DEST_SYNC          0x80
#define BOTH_SYNC          0xC0
#define SYNC_MODE          NO_SYNC
#define UNIFIED_MODE       0x00000 /* bit 14 */
#define SPLIT_MODE         0x4000
```

Recycling Interrupts

```
#define ENABLE_TCC_INT      0x40000 /* interrupt on complete */
                             /* bit 18 */

#define RESET_DMA          0x000000 /* bits 22-23 */
#define DMA_HALT_ON_RW_BOUNDARY  0x400000
#define DMA_HALT_ON_TRANSFER_BOUNDARY 0x800000
#define DMA_START          0xC00000

/* load control mode: stop when done, interrupt CPU on completion */
*(int *)DMA_CONTROL_REGISTER |= DMA_PRIORITY /* bits 0-1 */
    | TRANSFER_MODE /* bits 2-3 */
    | SYNC_MODE /* bits 6-7 */
    | UNIFIED_MODE /* bit 14 */
    | ENABLE_TCC_INT /* bit 18 */
    | DMA_HALT_ON_TRANSFER_BOUNDARY; /* bits 22-23 */

/* start DMA transfer */
*(int *)DMA_CONTROL_REGISTER |= DMA_START;
}
```

Synchronization

```
void front_end_initial_sync(void)
{
int n_tossed;
volatile int i;
volatile int i1, i2, i3, i4, i5, i6;
switch(whats_my_function())
{
case FFT_FRONTEND1: /* if this code is running on the FFT1 processor */
    reset_semaphore(SEM_FROM_FFT2_TO_FFT1S);
    /* so I can see if anyone is diddling it */

    signal(SEM_FROM_FFT1_TO_FFT2S);          /* I'm here, are you? */
    snooze(20*WAIT_LOOP_INTERVAL,usec);    /* wait for a response */

    /* check for a response */
    theres_a_second_frontend = (test_semaphore(SEM_FROM_FFT2_TO_FFT1S));
    snooze(10000,usec);
    hold_A_D();                            /* stop the A/D from filling FIFO */
}
```

Synchronization

```
if(theres_a_second_frontend)
{
    reset_semaphore(SEM_FROM_FFT2_TO_FFT1S); /*so we can now wait */
    signal(SEM_FROM_FFT1_TO_FFT2S); /* go ahead, drain your FIFO */
    n_tossed = drain_FIFO_nonblocking(DMA_INPUT_COMM_PORT1);
    i = drain_FIFO_nonblocking(DMA_INPUT_COMM_PORT1);
    if(i > 0)
    {
        printf("FIFO is not emptying: %d more samples found\n", i);
        exit(0);
    }
    *(volatile int *)N_TOSSED1 = n_tossed;

    /* I'm draining mine */

    /* wait for FFT2 to complete */
    short_wait(SEM_FROM_FFT2_TO_FFT1S, __LINE__);
    toggle_tclk1();
}
```

Synchronization

```
else
{
    /* if FFT1 is alone, it can just drain */
    n_tossed = drain_FIFO_nonblocking(DMA_INPUT_COMM_PORT1);
    i = drain_FIFO_nonblocking(DMA_INPUT_COMM_PORT1);
    if(i > 0)
    {
        printf("FIFO is not emptying: %d more samples found\n", i);
        exit(0);
    }
    *(volatile int *)N_TOSSED1 = n_tossed;
}
release_A_D(); /* and restart the A/D */
break;
```

Synchronization

```
/* note: to keep things simple, if there are two FFT_FEs, start #2 first */
case FFT_FRONTEND2: /* if this code is running on FFT2 */
    reset_semaphore(SEM_FROM_FFT1_TO_FFT2S); /* prepare to wait for FFT1 */
    wait(SEM_FROM_FFT1_TO_FFT2S, __LINE__); /* wait for FFT1 */
    signal(SEM_FROM_FFT2_TO_FFT1S); /* tell FFT1 "FFT2 is present" */
    wait(SEM_FROM_FFT1_TO_FFT2S, __LINE__); /* wait to drain FIFO */

n_tossed = drain_FIFO_nonblocking(DMA_INPUT_COMM_PORT2);

i = drain_FIFO_nonblocking(DMA_INPUT_COMM_PORT2);

if(i > 0)
{
    printf("FIFO is not emptying: %d more samples found\n", i);
    exit(0);
}
```

Synchronization

```
*(volatile int *)N_TOSSED2 = n_tossed;
snooze(1000,usec); /* make sure FFT1 had time to finish */
signal(SEM_FROM_FFT2_TO_FFT1S); /* indicate that FIFO is done */
toggle_tclk1();
break;
default:
printf("in front_end_initial_sync(), invalid whats_my_function() = %d\n",
      whats_my_function());
exit(1);
}
}
```

Synchronization Primitives

```
void set_semaphore(int address)
{
  *(volatile int *)address = TRUE;
}
```

```
void reset_semaphore(int address)
{
  *(volatile int *)address = FALSE;
}
```

```
int test_semaphore(int address)
{
  return(*(volatile int *)address);
}
```

Synchronization Primitives

```
void wait(int address, int line)
{
    volatile int count=0;
    while( !*(volatile int *)address )
    {
        snooze(WAIT_LOOP_INTERVAL, usec);
        if(++count > 10000000)
        {
            printf("excessive wait for 0x%8x at line %d. process quits\n",
                    address, line);
            exit(0);
        }
    }
    *(volatile int *)address = FALSE;
}
```

```
void signal(int address)
{
    *(volatile int *)address = TRUE;
}
```

C-Callable Assembler

```
.version 40
FP      .set          AR3

        .globl      _fft_asm
*****
* FUNCTION DEF : _fft
*****

_fft_asm:
        PUSH      FP
        LDI       SP,FP
        PUSH      R4
        PUSH      R5
        PUSHF     R6
        PUSHF     R7
        PUSH      AR4
        PUSH      AR5
        PUSH      AR6
        PUSH      AR7
        PUSH      R8
```

C-Callable Assembler

```
*** ----- ; Read arguments off stack
LDA    *-FP(2),AR7      ; cmplx_in
LDA    *-FP(3),AR5      ; sin_ptr
LDI    *-FP(4),R9       ; N
LDI    *-FP(5),R8       ; log2N
LDI    *-FP(6),R2       ; off_chip_array

*** ----- ; if ( N <= 0 ) goto g4;
CMPI   0,R9
BLE    L14              ;

L14:
*** ----- ; if ( (C$10 = N*2) <= 0 ) goto g8;
LSH    1,R9,R10
BLE    L18

*** ----- ; Transfer complex array from off_chip_array
*** ----- ; location to cmplx_in using bit-reversed
*** ----- ; addressing
LDI    R9, IR0          ; Required for bit-reversed addressing
LDA    R2,AR2           ; AR2 = off_chip_array
```

C-Callable Assembler

```
*** ----- ; Execute loop N times
SUBI    1,R9,RC
RPTBD   SHUFF ----- ; Delayed loop start
*** ----- ; AR4 = cmplx_in;
LDI     2, IR1
LDA     AR7,AR4
LDF     *+AR2(1), R0 ----- ; Fetch Q sample
*** ----- ; Start of loop
LDF     *AR2++(IR0)B, R0 ; Fetch I sample, bit-reversed increment
|| STF R0, *+AR4(1) ----- ; to next location
SHUFF:
LDF     *+AR2(1), R0 ----- ; Fetch next Q sample
|| STF R0, *AR4++(IR1) ----- ; Store I sample, normal increment
```

C-Callable Assembler

L18:

```
*** ----- ; Input data is now in array cmplx_in
*** ----- ; in shuffled order : xr(0), xi(0), xr(N/2),
*** ----- ; xi(N/2), ..etc
*** ----- ; The 1st stage of FFT has twiddle factors
*** ----- ; 1 and -1, so a separate loop is used.
*** ----- ; Loop setup
LDA     AR7,AR2           ; AR2 = temp_ptr = cmplx_in
ADDI    2,AR2,AR4        ; AR4 = temp_ptr2 = &temp_ptr[2];
ASH     -1, R9, R1       ; Execute loop N/2 times

ADDI    -1, R1, RC

*** -----
RPTBD   FIRST_LOOP      ; Delayed loop start
*** -----
LDI     3, IR0
LDF     *AR4, R5         ; bi = temp_ptr2[0]
|| LDF  *+AR4(1),R3      ; bj = temp_ptr2[1]
*** -----
```

C-Callable Assembler

```
*** -----
SUBF      R5,*AR2,R2      ; s1 = ai - bi
*** ----- ; START OF FIRST_LOOP
ADDF      R5,*AR2++,R4    ; s3 = ai + bi
|| STF    R2, *AR4++      ; bi = s1
*** -----
SUBF      R3,*AR2,R2      ; s2 = aj - bj
|| STF    R4, *-AR2(1)    ; ai = s3
*** -----
ADDF      R3, *AR2++(IR0),R2 ; s4 = aj + bj
|| STF    R2, *AR4++(IR0) ; bj = s2
*** -----
LDF      *AR4, R5        ; bi = temp_ptr2[0]
|| LDF    *+AR4(1),R3     ; bj = temp_ptr2[1]
*** -----
FIRST_LOOP:
STF      R2, *-AR2(IR0)   ; aj = s4
|| SUBF   R5,*AR2,R2      ; s1 = ai - bi
*** ----- ; End of FIRST_LOOP
```

C-Callable Assembler

```
***      80 more lines of the same stuff...
***      -----
***      ----- ; CACHE_FREEZE() asm("
OR  0400H,ST      ; Freeze cache
***      ----- ; Cache freeze")
***      ----- ; return;
```

EPI0_2:

```
      LDI      *-FP(1),R1
      LDI      *FP,FP
      POP      R8
      POP      AR7
      POP      AR6
      POP      AR5
      POP      AR4
      POPF     R7
      POPF     R6
      BD       R1
      POP      R5
      POP      R4
      SUBI     2,SP
***     B       R1      ;BRANCH OCCURS
```

C-Callable Assembler

```
*** ----- ; The remaining log2(N) - 1 FFT stages
*** ----- ; are executed in 3 nested loops
*** ----- ; L31, L32, L33 with loop counts
*** ----- ; log2N-1, set_no and samp_offsetby2.
*** ----- ; With every execution of outermost loop
*** ----- ; L31, set_no decreases from N/4 to 1
*** ----- ; and samp_offsetby2 increases from 2 to N/2
*** ----- ; LOOP setup
ASH    -2,R9,BK    ; cosi_ptr = &sin_ptr[(N>>2)];
ADDI   BK,AR5,AR6
LDI   AR5, R7      ; R7 = sin_ptr
*** -----
ASH    -2,R9,R9    ; R9 = set_no = N>>2
SUBI   2,R8,R8     ; R8, L31 loop counter = log2N-2;
LDI   4,R11       ; samp_offset = 4;
```

Source code examples

- Complete listings will be posted on WebCT, if you are interested

Homework #7

- Find an implementation of the FFT in C (e.g., Embry published a book on Signal Processing with C that includes FFT routines)
- Generate 1000 sets of 1024 random complex values and compute their FFT, measuring the time needed to calculate the FFT (not random number generation)
- Perform the same measurement using Matlab and the built-in FFT function.
- Compare your results.
- Optional: optimize the C function.