Lab #1

Lab #1 at 10.20am Thu, Feb. 3 and 2.30pm Fri Feb. 4.

General Overview of Modified PASS.ASM for the DSP56302EVM

In this section we provide a general overview of **`pass.asm`** for the DSP56302EVM. This overview will be similar to **`pass.asm`** for other DSP5630*x*EVMs. See Appendix B in textbook for code listing.

Lines 1 - 23 At the top of **`pass.asm`** we place a descriptive header which is to be filled out for each project. In this header is an area to define variables and associated addresses. You are strongly encouraged to carefully document your code in the header..

Line 26 Includes the file **`ioequ.asm`** which contains equate statements for the DSP56302's I/O registers and ports. All of the mnemonics and values can be found in the *DSP56302 User's Manual*.

Line 27 Includes the file **`intequ.asm`** which contains equate statements for the DSP56302 interrupts. All of the mnemonics and values can be found in the *DSP56302 User's Manual*. This file also establishes the base address for the interrupt vector table, `I_VEC`.

Line 28 Includes the file **`ada_equ.asm`** which initializes constants to facilitate initialization of the codec. Complete information regarding these constants can be found in the *Crystal Semiconductor CS4215 Data Sheet*.

Line 29 Includes the file **`vectors.asm`** which lists instructions in the vector table to execute upon interrupt. We will be mostly concerned with interrupts generated with putting/getting samples to/from the codec. Instructions associated with codec interrupts and found on Lines 84 – 94 in **`vectors.asm`** are:

```
jsr ssi_rx_isr   ;- ESSI0 Receive Data
jsr ssi_rxe_isr  ;- ESSI0 Receive Data w/ Exception Status
jsr ssi_rxls_isr ;- ESSI0 Receive last slot
jsr ssi_tx_isr   ;- ESSI0 Transmit Data
jsr ssi_txe_isr  ;- ESSI0 Transmit Data w/ Exception Status
jsr ssi_txls_isr ;- ESSI0 Transmit last slot
```

Code for the interrupt service (sub)routines (ISRs), `ssi_rx_isr`, `ssi_tx_isr`, etc... can be found in **`ada_init.asm`** which we will discuss shortly.

Line 35 We include a file called **`pass.dat`** which is described shortly.

Line 40 Instructs the assembler to place our code in program memory beginning at address p:$100.

Lines 39 - 48 Establish some basic operational parameters of the DSP including clock speed, memory wait states, interrupt masks, initialization of the hardware stack, operating modes, and software stack pointer.

External memory. In order to access external memory some steps must be taken. These steps are outlined in Section 4.3 in the *DSP56302EVM User's Manual* and assume mask identification numbers (middle line on the DSP56302 part) of F90S and higher. These parts include an address priority disable (APD). We define Address Attribute Registers (AARs) for areas 0 and 3 in **pass.dat** so that the external memory is partitioned into two 16K banks for **x** and **y** memory and the Address Attribute 3 (AA3) pin acts as a switch, toggling between the upper and lower halves of external memory. These partitions each have a base address of **$010000**. This implies that the only available memory is on-chip between addresses **$000000**--**$001BFF** (assuming the default memory map SC=0, MS=0, and CE=0 in the OMR) and off-chip between addresses $010000--$013FFF. Memory addresses

between $001C00--$00FFFF are unavailable.

DSP56302EVM Memory Map

Address	X-Memory	Y-Memory
$013FFF	off-chip (16K)	off-chip (16K)
$010000		
$00FFFF	Unavailable	Unavailable
$001C00		
$001BFF	on-chip (7K)	on-chip (7K)
$00000A		
$000009	Reserved	on-chip
$000000		

To complete the external memory configuration we set OMR bits to 0 except for bit 14 which is set to 1. Among other things, this configuration sets the default on-chip memory map to 20K of program RAM (p memory), 7K of X data RAM (x memory), and 7K of Y data RAM (y memory). Bit 14 is set to enable the address priority disable. Finally, we must physically set the jumper, J9 on the DSP56302EVM to connect pins 1 and 2.

Software Stack. We note that the address register, r6 is used to point to the software stack. The ISRs use the stack in their operation and thus you must *never* use r6 except for stack operations. Misusing r6 will cause the ISRs to not perform correctly.

Line 50 Executes the subroutine **`ada_init`** (contained in the included file **`ada_init.asm`**). This subroutine initializes the codec using some of the control words defined in **`pass.dat`**.

Lines 52 - 54 Depending on the value of `ADA_OFF`, Line 53 (which masks interrupts and disables the codec) is conditionally assembled into the code. This will be useful for debugging code. `ADA_OFF` is defined in **`pass.dat`**.

Line 56 We initialize our program with the subroutine **`proginit.asm`**.

Lines 58 - 64 Final initialization of the codec includes moving additional control words to the codec as well as clearing storage for received sample values.

Lines 66 - 77 form the main event loop. We begin with the label, **main_loop** defining the beginning of the main event loop. In this loop we receive right and left samples from the codec, execute the sample processing subroutine **process_stereo**, and transmit right left output samples. We then return to the beginning of the main event loop.

The first two instructions in **main_loop**, jset determine the beginning of a new sample period. While waiting for a "frame sync to pass," interrupts are generated to get/put samples and control words from/to the codec. The ISRs are executed in response to these various interrupts.

Line 78 Includes the file **`ada_init.asm`** which contains the codec initialization routine executed in Line 113 as well as the ISRs.

Lines 79 - 81 We include three files called **ada_init.asm**, **proginit.asm** and **procster.asm**. The file

ada_init.asm contains the codec initialization routine executed in Line 50 as well as the ISRs. The file **proginit.asm** contains the initialization routine executed in Line 56 and the file **procster.asm** contains the student-written signal processing subroutine called in Line 73.

Note that you may place code (subroutines) in separate files to aid in code organization. In addition there may be several subroutines within any file. If written properly, these files can be reused in other projects. Normally these files are placed after line 81.

DSP5630*x* Clock Speed

The DSP56300 core features a phase-locked loop (PLL) clock oscillator in its central processing module. The PLL allows the processor to operate at a high internal clock frequency using a low frequency clock input. This feature offers two immediate benefits: lower frequency clock input reduces the overall electromagnetic interference generated by the system and the ability to oscillate at different frequencies reduces costs by eliminating the need to add additional oscillators to a system.

The DSP56302EVM has a 16MHz clock which can be multiplied internally to operate the DSP at rates higher than 16MHz. The DSP56302 on the EVM is rated at 66MHz which implies that we need to multiply the PLL by a factor of 4. Overclocking the DSP beyond its rated clock speed could result in serious damage to the part. Line 39 in **pass.asm**

```
movep #$040003,x:M_PCTL ; set PLL for MPY of 4X
```

performs this PLL multiplication function. The PLL control register (PCTL) directs the operation of the on-chip PLL. The PCTL control bits are defined in Section 9.2.3 of the *DSP56300 Family Manual*. Several of these control bits are the multiplication factor bits which will multiply the input clock frequency according to Table 9-1 in *DSP56300 Family Manual*.

PASS.DAT

We describe the **`pass.dat`** file defines labels used in the program and lays out data memory.

Line 8 Establishes the `ADA_OFF` variable used to disable the codec. When this variable is set to 1, code (lines 71 - 73 in **`pass.dat`**) which masks interrupts is conditionally assembled.

Lines 9 – 10 Define the AARs required for our configuration of external memory.

Line 41 Defines the starting point where the user can lay out data in 7K of internal X-memory. We note the lower address of $00000a since the first ten ($000000 – $000009) words in X-memory are used by the codec and are thus reserved. The upper address for internal X-memory is $001BFF.

Line 59 Defines the starting point where the user can lay out 16K of external X-memory. We note a lower address of $010000 and an upper address of $013FFF. Addresses $001C00 – $00FFFF are unavailable.

Line 65 Defines the starting point where the user can lay out 7K of internal Y-memory. We note a lower address of $000000 and an upper address of $001BFF.

Line 70 Defines the starting point where the user can lay out 16K of external Y-memory. We note a lower address of $010000 and an upper address of $013FFF. Addresses $001C00 – $00FFFF are unavailable.

PROGINIT.ASM

We describe the **`proginit.asm`** file contains developer initialization code such as that used to clear or initialize memory.

PROCSTER.ASM

We describe the **`procster.asm`** file contains the code for the actual signal processing.

ADA_EQU.ASM

The **`ada_equ.asm`** file contains labels to facilitate codec initialization and operation. One modification that developers may wish to make is the sample rate of the codec. Control words for the various sample rates that the codec is capable of are defined in this file. These lines read

```
      SAMP_RATE_48 equ $003000
.
.
.
      SAMP_RATE_8 equ $000000
```

In order to change the sample rate using one of the defined control words, we need to edit the line

```
CTRL_WD_12  equ   NO_PREAMP+HI_PASS_FILT+SAMP_RATE_48+STEREO+DATA_16 ;CLB=0
```

by replacing `SAMP_RATE_48` (f_s = 48 kHz) with another sample rate control word. The available control words are

`SAMP_RATE_32` (f_s = 32 kHz)
`SAMP_RATE_27` (f_s = 27 kHz)
`SAMP_RATE_16` (f_s = 16 kHz)
`SAMP_RATE_96` (f_s = 9.6 kHz)
`SAMP_RATE_8` (f_s = 8 kHz)

Then assemble, download, and run.

Get familiar with the sounds as you alter code. Clearly the lower the sampling rate, the lower the bandwidth of the signal (according to Nyquist theory). One of the most important lessons you will learn from this course is evaluating code from the sounds it produces.

Program #5 (FIR Filter Example) (Done in Lab #2)

This first simple program filters the right channel with a 10 point Moving Average (MA) FIR filter. We develop the code in three steps:

Step 1: Allocate vectors (arrays) in memory to store samples and coefficients
Step 2: Initialize address registers (pointers) to samples and coefficients. Clear out the input sample vector
Step 3: Place the FIR filter code in **`procster.asm`**
Step 4: Transmit the output

Step 1. In **`PASS.DAT`** add the following lines in the appropriate places:

```
N     equ   10    ;length of circular queue

      ORG   x:$00000a
RQUEUE      dsm   N     ;malloc N words for right channel input samples

      ORG   y:$000000
RCOEFS      dsm   N     ;malloc N words for right channel coeffs

      ORG   y:RCOEFS
      dc    0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1   ;simple MA filter
```

Step 2. Replace the NOP in **PROGINIT.ASM** with the following lines:

```
;Initialize pointers to queues and coefficients, set up modulo addressing
      move  #RQUEUE,r0  ;point to right channel sample queue
      move  #N-1,m0     ;set up modulo addressing for right queue
      move  #RCOEFS,r5  ;point to right FIR coefficients
      move  #N-1,m5     ;set up modulo addressing for right coefficients
      move #$0,x0       ;now clear out input queue
      rep   #N
      move  x0,x:(r0)+
```

Step 3. Replace the NOPs in **PROCSTER.ASM** with FIR code:

```
      clr   a     x0,x:(r0)+  y:(r5)+,y1  ;repl old R samp w/current,get 1st coef
      rep   #N-1
      mac   x0,y1,a     x:(r0)+,x0  y:(r5)+,y1
      macr  x0,y1,a     (r0)-
```

Step 4. Assuming the right filter output is stored in accumulator a and the unfiltered left sample is passed through, make the following changes to the associated lines in the main event loop

```
      move a,x:TX_BUFF_BASE    ;transmit right
      move y0,x:TX_BUFF_BASE+1 ;transmit left
```

