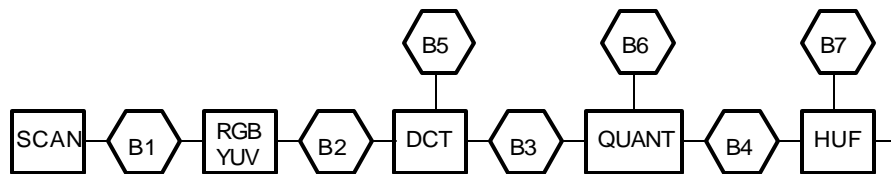


FIXED POINT REFINEMENT OF THE JPEG ENCODER

When the JPEG Encoder is implemented on an embedded platform, we want to reduce the operation word-lengths as much as possible. This reduces the computational cost of the operations drastically – and consequently also their energy consumption. For hardware implementation, we can choose customized word-lengths for each operation, and design operators that have just the right word-length to accommodate all time-multiplexed operations they implement. For DSP (software) implementation, the word-length is fixed by the choice of processor: 16-bit, 32-bit and so on but still requires the software description to be expressed in terms of the target word-length.

Fixed point refinement requires quantization of signals and is a non-linear operation. The optimal choice of word-lengths therefore is a hard problem. Typically it is solved by combining simulation with analytic methods. For example, when adding two 10-bit numbers, we know that the result can be captured in 11-bit precision (analysis). Out of simulation, we can however conclude that the 11 bits of precision are not used in 99.99% of the cases, and that consequently we can use only 10 bits of output precision in combination with an overflow handling strategy (simulation).

In the case of the JPEG Encoder, we are lucky in the sense that we start and end with well-defined wordlengths. A pixel is 8-bit R data, 8-bit G data and 8-bit B data. The use of floating point in the encoder is limited to constant multiplications and variable additions. Considering the reference implementation, we only find floating point numbers in RGBYUV, DCT and QUANT. Moreover, as we have designed a system model earlier (HW3), we have a partitioned model available that separates the JPEG Encoder into individual functions that can be quantized separately.



On the project homepage you will find a dataflow model of the JPEG encoder in C (Solution to HW2). This model implements the same functionality as the reference C++ code, but has been partitioned according to the idea of HW3. The dataflow model closely reflects the system model shown in the figure. Each block is captured in a separate C file. The communication channels between the blocks (dataflow queues) are implemented with a type called `FIFO`. All channels are point-to-point and transport tokens of type `short` (16-bit). Each block is implemented by means of two functions: an init function called to perform initialization, and a run function to perform computation according to the dataflow computation model. When run is called, the firing rule of the block is tested first. This test simply checks the number of available tokens on the input FIFO's connected to that block. When sufficient tokens are available, the block can fire (compute) and produce output tokens. The dataflow schedule is implemented in the main function.

In this homework, you will do fixed point refinement for your assigned platform. Check the project homepage to find out which that one is.

- If your target is TI C54 or Analog Devices Blackfin, you will do refinement to 16-bit fixed-point DSP. This implies you will turn in a C-program of a JPEG encoder that only uses `short`.
- If your target is Handle-C you will do refinement for hardware. In each block, you can use a difference wordlength for each block.
- If your target is ART designer, you will do refinement for ASIP. This means that you can use customized word-lengths for an operation, but you should keep in mind that the choice is not arbitrary because of the time-multiplexing of operations in the ART datapath. That is, for each new, unique wordlength you introduce, you will introduce extra type alignment hardware (multiplexers) at the input of functional units.

THE ASSIGNMENT IF YOUR PLATFORM IS C54 OR BLACKFIN

For this assignment, you need to focus your attention to three blocks: RGBYUV, DCT, and QUANT. In each of the C implementations for these blocks (rgbyuv.c, dct.c, quant.c), you will find that there are floating point numbers in use. Your job is to rewrite those functions such that they no longer need floating point. When you introduce new types in the C-code, keep in mind the following required precisions:

- int, unsigned int, long, unsigned long are 32 bit
- char, unsigned char are 8 bit
- short, unsigned short are 16 bit

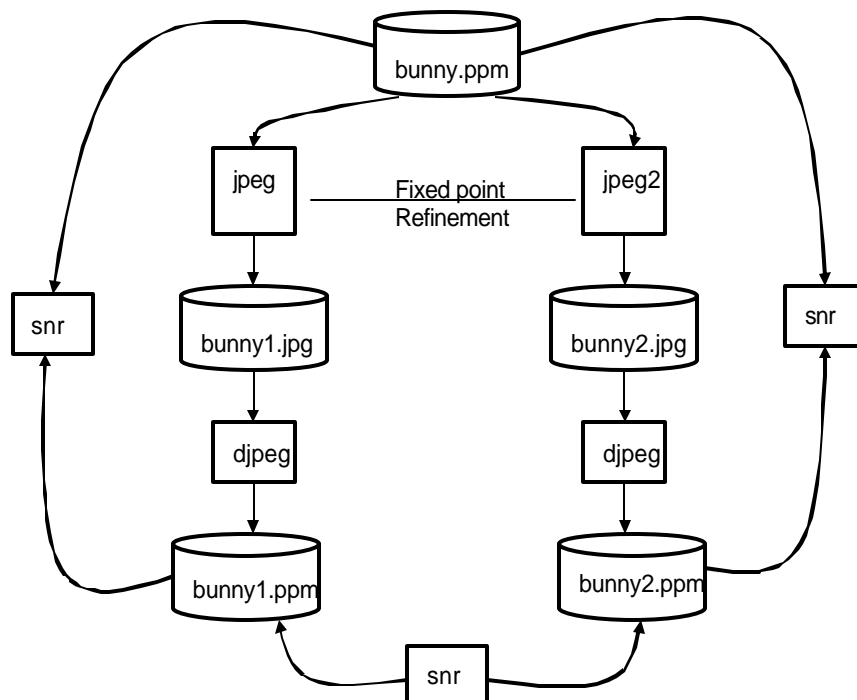
Skip to section “quality monitoring” for the rest of the assignment.

THE ASSIGNMENT IF YOUR PLATFORM IS ART DESIGNER OR HANDLE C

For this assignment, you need to focus your attention to three blocks: RGBYUV, DCT, and QUANT. In each of the C implementations for these blocks (rgbyuv.c, dct.c, quant.c), you will find that there are floating point numbers in use. Your job is to rewrite those functions such that they no longer need floating point. When you introduce new types in the C-code, you will make use of the ART designer fixed point library. This library can be found on EEnet, /usr/apps/art/{sol,hp}/lib. The library allows you to use fixed point types in your code.

QUALITY MONITORING

While doing quantization and fixed point modeling, you will have to monitor image quality in a formal way. A small program that compares two PPM images is provided for that. The program, snr.cxx, calculates the sum of squares of pixel differences between two images. The JPEG images that are created by your simulation need first to be converted to PPM format. You can use the standard UNIX utility djpeg for that.



As shown in this figure, you can produce three values with snr:

- Compare the original input image (ppm) with the image created using the original simulation
- Compare the original input image with the image created using the fixed-point refined simulation
- Compare the resulting images created using the original simulation and the fixed-point refined simulation

While doing fixed-point refinement, your goal is to **keep the degradation introduced by fixed point refinement 5 dB under the degradation introduced by JPEG compression**. It is up to you to figure out how to translate this spec to the design flow shown in the figure.

HOW TO START

Download the package for HW4 from the ee201a project homepage. You will find the following subdirectories and code contained within it:

- `jpeg_df`: The jpeg dataflow model which is used as a starting point for fixed-point refinement for a 16-bit DSP processor (C54 or BlackFin).
- `jpeg_df_art`: The jpeg dataflow model which is used as a starting point for fixed-point refinement for Handle-C and ART Designer. This contains the same code as under `jpeg_df`, only the linker command has been slightly modified to use the ART Lib libraries. When you use these libraries, you must be logged into an HP station from the EE lab with `g++ 2.95`. (This includes `hplab01.ee.ucla.edu` to `hplab17.ee.ucla.edu`)
- `snr`: The program that compares two PPM images

Note that `djpeg` is a UNIX command line utility already available. Using the programs above, you can create a design flow as shown in the figure. The '`jpeg2`' program is the only program that you need to provide. You create it by first making a copy of `jpeg_df` or `jpeg_df_start` and next incrementally modifying it.

WHAT TO TURN IN

Due Date: 5/9/2002 at 12 o'clock noon at Letty's desk (7440 Boelter Hall).

One sheet containing the following:

- Your name
- Link to your homepage where you post the resulting, fixed-point refined C program. Do not provide public access to this page before 5/9/2002 12 o'clock.
- The three SNR figures as obtained from the design flow indicated above. Indicate which SNR figures you have to compare to demonstrate the 5 dB bound indicated above.