A Multiple DSP-based 3D Laser Range Sensor and its Application to Real-time Motion Detection

Green, David; Blais, François

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

NRC·CNRC

*A Multiple DSP-based 3D Laser Range Sensor and its Application to Real-time Motion Detection.**

Green, D., and Blais, F.

May 2002

* published in: NRC/ERB-1095. May 2002, 201 pages.

Canada

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

ERB-1095

A Multiple DSP-based 3D Laser Range Sensor and its Application to Real-time Motion Detection.

Green, D., and Blais, F.
May 2002

NRC 44930

Table of Contents

List of Figures

A Multiple DSP-based 3D Laser Range Sensor and its Application to Real-time Motion Detection

David Green, Francois Blais
Institute for Information Technology
National Research Council of Canada

Abstract

This report describes the implementation of an auto-synchronised 3D laser range scanner and its application to motion detection. The 3D motion detector (3DMD) uses a 3D laser scanner to detect the motion of an object and to display that motion in real-time. The system is based on a real-time modular digital signal processor (DSP) architecture called MDSP. 3DMD is a PC-based system that contains an array of embedded DSP processors that communicate using dedicated communication links. The scalable DSP system is based on a message communication model, rather than a shared memory model. The software is based on Precise/MQX, a commercial real-time operating system. The report describes the principle of operation of 3DMD and describes in detail the hardware and software configuration. The report includes some practical information about how to load and run the system demonstration, and how to develop application software for 3DMD. It concludes with some preliminary performance results.

1. Introduction

To date, NRC auto-synchronised laser scanners[1] have been used in many applications where precise range measurements are required. Such applications have included the measurement and capture of complex surfaces associated with museum artefacts, the human body, manufactured products and of even large objects such as orbital space platforms[2,3,4,5]. This report describes a new experimental implementation of an auto-synchronised scanner and its application to motion detection. The system described here uses a 3D scanner to detect the motion of an object within a specified field of view and to display that motion in real-time. When fully calibrated, this motion detection system will be able to provide quantitative measurements of object motion.

An example of a possible application of this technology could include measuring the chest movement of a breathing patient in order to position and synchronise medical diagnostic equipment. Other applications might include the measurement of head motion for use in the presentation of virtual environments.

The 3D motion detector (3DMD) uses an auto-synchronised scanner to capture real-time range data over a predetermined volume. The scanner is an active device, which sweeps a laser beam across the surface being captured, and collects range and intensity information about the target in the process. In previous applications, these data are used to generate surface models and ultimately 3D models of scanned objects. While arbitrary scanning waveforms can be used, we have chosen the Lissajous pattern in order to optimise performance[6]. It achieves a high scan rate at the expense of a lower spatial sampling density. Typically, each scan line is composed of 512, 1024 or 2048 range/intensity samples.

The calculation of motion detection is based on the scan-by-scan analysis of range data. At the completion of each scan 'line', a smoothed running average of the range associated with each sample point is updated. Then the departure of each sample point from the smoothed average is computed. These departures are then compared to a threshold. If any point exceeds that threshold, its co-ordinates are saved in an output motion buffer. An object moving in any direction within the operating envelope of the system will be detected.

The system that has been implemented is based on the MDSP architecture[7]. This is a PC-based system that includes an array of embedded DSP processors which not only implement the laser range scanner itself but also perform the follow-on processing required for motion detection. The embedded DSP system accepts commands from the user via a graphical user interface on the PC. The PC is also used for the graphical presentation of motion results.

This report begins with a description of the principle of operation of the 3DMD. This summarises the components of the system and will also outline the processing that is performed for motion detection. Section 3 continues with a description of the user interface for the prototype system. Sections 4 and 5 describe the hardware and software configuration of the laser scanner system used for 3DMD. The system is built using an embedded array of DSPs which communicate using dedicated communication links. The scalable system is based on a message communication model, rather than a shared memory model. The software is based on MQX, a commercial multiprocessor real-time operating system[8]. Section 6 explains how to actually load and run the demonstration. Section 7 discusses some preliminary performance results and the final section concludes and summarises the report. The appendix includes detailed software listings and a design note discussing the installation and initial port of MQX to the DSP hardware.

2. Principle of Operation

2.1. Scanner Operation for Motion Detection

Details of the operation of the 3D auto-synchronised scanner are beyond the scope of this report but are available from other sources[1]. In summary, the scanner Fig. 1 emits a beam of laser light, which is scanned using computer controlled deflection mirrors through a

specified field of view. Measurement of range is based on optical triangulation whereby target range is computed as a function of the precise position of the returned laser spot on the CCD photodetector. The scanner generates range and intensity samples as the laser beam strikes any diffusing surface that lies within its depth-of-field. In this prototype system, the range data are uncalibrated; the use of simple calibration in real-time would yield quantitative range data.

The scanner used for 3DMD is a 2-axis auto-synchronised scanner in which laser pointing is controlled by two independent scanning mechanisms. A Lissajous scan pattern is used whereby each axis is driven by a sinusoidal waveform. By controlling the amplitude, frequency and phase of the 2 waveforms, a variety of Lissajous patterns can be generated. Fig. 2 shows examples of 3 x 4 and 5 x 6 Lissajous patterns.

Range and intensity data are collected in real-time as each scan progresses. A Lissajous pattern is used because it can cover a large fraction of the field-of-view of the sensor at high speed. While the coverage is sparse, it is suitable for motion detection purposes. Although a raster scan would offer much more complete coverage, it would be significantly slower.

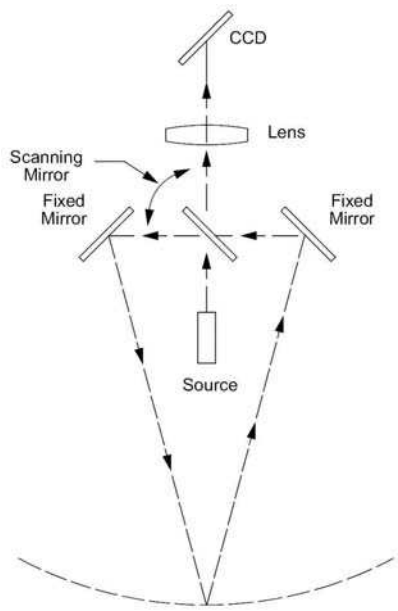

Figure 1. Basic Configuration of an Auto-synchronised Scanner.

Optical triangulation on its own provides high resolution but limited field-of-view. The addition of an oscillating double-sided mirror scans the optical path to produce a large field-of-view.

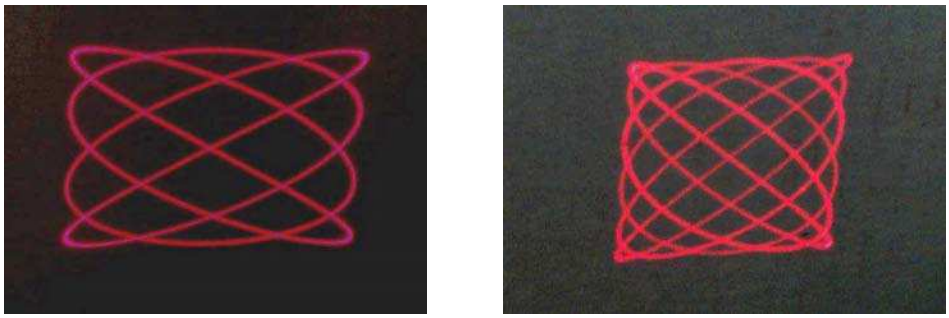

Figure 2. Examples of 3 x 4 and 5 x 6 Lissajous Scanning Waveforms.

2.2. Motion Detection

During operation, the scanner repeatedly monitors a fixed volume in space. Because of the stability of the scanning mechanism, data samples remain angularly registered from one scan to the next and can therefore be compared on a voxel-by-voxel basis. If one of several objects within the scanner's field-of-view is in motion, then sequential range samples obtained from the surface of the moving object will show a variation in range. By calculating a 'moving average' using a smoothing IIR filter, it is possible to identify those points along the scan that are in motion. A moving average calculation, Equation 1, is applied to each point along the scan.

$$R_i(t) = (1-\alpha)\ ?R_i(t-1) + \alpha\ ?I_i(t) \tag{1}$$

Where $R_i(t)$ is the smoothed range value for the ith data sample at time t and $I_i(t)$ is the current range sample. The parameter ‘α’ defines the time constant of the smoothing filter. For small values of α, the smoothed range value responds very slowly to changing range data. Points in motion can be detected by subtracting the current range value from the average value for that point as shown in Equation 2.

$$D_i(t) = R_i(t) - I_i(t) \tag{2}$$

If a point is in motion, then the difference value $D_i(t)$ will be driven either positive or negative, depending on whether the direction of motion is towards or away from the scanner. After a moving point comes to rest, $D_i(t)$ will decay to 0 at a rate determined by α. A threshold is used to eliminate noise.

2.3. Data Presentation

Each sample point is uniquely identified by its 'index' i.e. its position along the scan. When a point is determined to exceed the motion threshold, its index value and its corresponding range value are saved in a buffer.

Presentation of the motion data has taken two approaches. The first method uses the fact that the position along a scan of each range sample is known precisely. As a range sample corresponding to a point in motion arrives, its index value can be associated with a precise position along each of the two scanning waveforms. Therefore its position on a graphical display can be computed. In practice, it is also necessary to offset the computed position value of each axis by a phase shift introduced by the delay of the actual scanning mechanism, a galvanometer. These offsets, which are a function of the scan parameters, have been measured. They are incorporated directly into the software as 'offset corrections'.

The second method used for data presentation uses the actual galvanometer position data that are captured during each scan. These data can be used directly to determine the XY position of each motion point on the display.

3. The User Interface for the Demonstration System

A simple PC-based user interface for 3DMD has been implemented with Visual Basic. The prototype interface consists of a number of Visual Basic forms. On initial startup, the user encounters the “Waveform Controller” form, which is the central control form for the demonstration. It contains four objects: the Wave Generator control frame, the Main control frame, the target response display, and a picture box for display of the motion data. This form also has five pull-down menus: Waveform List, Waveform Adjust, Motion Parameters, Display Parameters and Help. Section 3 describes each form in detail. Section 6 describes in detail how to load and run the demonstration.

3.1 The Waveform Controller Form

On initial startup, the user encounters the “Waveform Controller” form Fig. 3. Startup synchronisation is required between the PC and the embedded DSP multiprocessor. Clicking the “Start Target” button generates a message in the Target Response display box indicating that initialisation is underway. When the “Waveform controller ready” announcement appears in the Target Response display, the “Init Target” button is enabled, all other controls remain greyed out. Clicking this button initialises the system. Once initialised, the “Show Pins” and “Quit” buttons are enabled. Also, “Go” and “Stop” buttons are enabled on the Wave Generator frame. The system is now ready to acquire and display data.

The system will start when the “Go” button is pressed and clicking the “Stop” button will stop the scanner immediately. The system can be demonstrated by placing a small target in the field-of-view of the scanner. The position of the target will be displayed in the display picture box. The bar graph to the right of the display picture box displays the average range to the target. All of these data are constantly updated. The default scan pattern used at startup is a 5 x 6 (H x V) Lissajous pattern consisting of 2048 points per scan.

The “Show Pins” button on the Main control frame is used for debugging. It lists the pinouts of a number of internal signals that can be monitored with an oscilloscope.

The following subsections describe the pull-down menus on the Waveform Controller form.

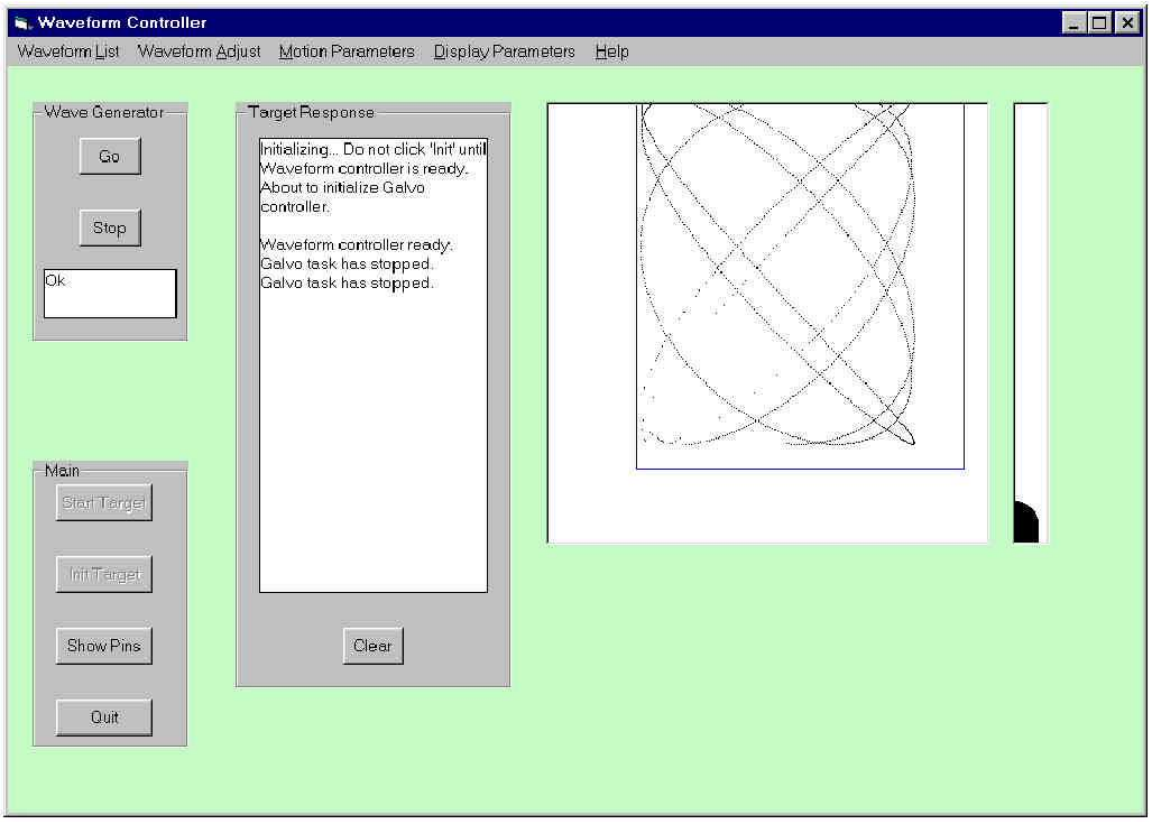

Figure 3. Waveform Controller Form.

This is the main user interface for 3DMD.

3.2 Waveform List

Selecting the Waveform List menu activates the Select/Edit Waveform form Fig. 4. The user can define and select up to eight scanning waveforms. The parameters of each waveform are listed on the form and can be adjusted. For each axis these include:

- the waveform type (COS or SAW) for cosine or sawtooth waveforms,
- amplitude in volts,
- phase in degrees,
- phase correction in degrees to compensate for the different inertia/speed of the two-axis galvanometers/mirrors ,
- offset in volts,
- points, the number of samples per scan. X and Y axes must be identical,
- cycles, the number of cycles within a single scan,
- Fast axis, 'Yes/No' selects which axis is updated at the sample rate, the other will be updated at the 'line rate'. This allows either horizontal or vertical rasters to be generated.

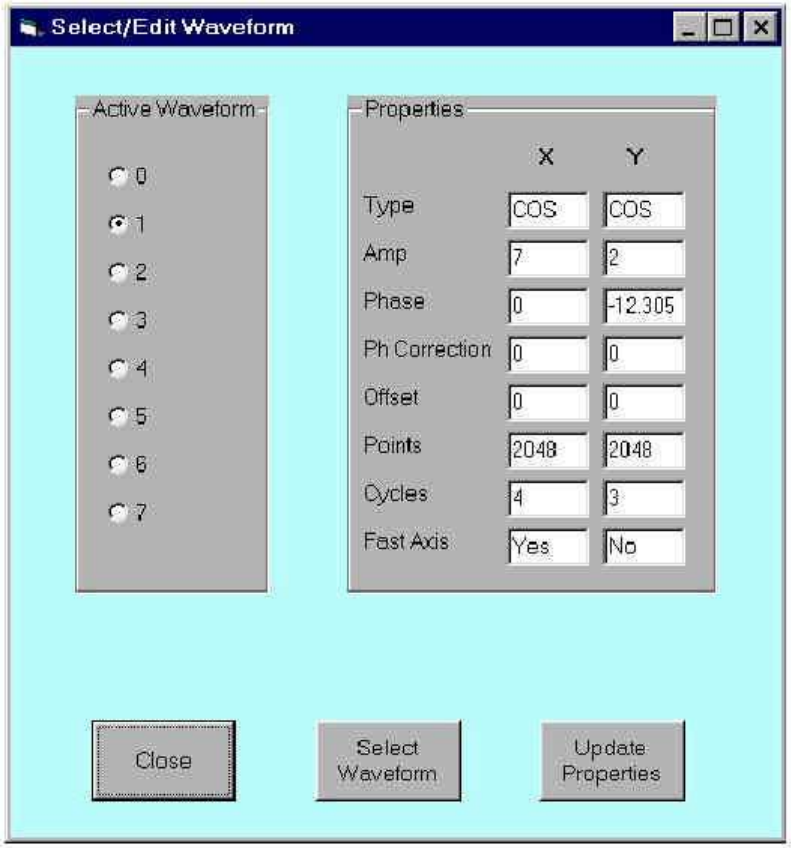

Figure 4. Select/Edit Waveform Form.

This form allows as many as 8 waveforms to be configured. Waveform selections can occur at runtime but waveform updates can only be made while the scanner is idling.

Once the waveforms have been defined, they can be selected dynamically at runtime by choosing the appropriate option button and clicking 'Select Waveform'. Waveform properties can only be edited while the scanner is idle.

Mathematically, a Lissajous waveform pattern is defined as:

$$x_i = A_x \cos\left(Cycles_x \cdot \frac{2 \cdot \pi \cdot i}{NbPoints} + Phase_x + Ph_x \right) + Offset_x \tag{3}$$

$$y_i = A_y \cos\left(Cycles_y \cdot \frac{2 \cdot \pi \cdot i}{NbPoints} + Phase_y + Ph_y \right) + Offset_y \tag{4}$$

3.3 Waveform Adjust

The "Waveform Adjust" form, Fig. 5 allows the scanning pattern to be adjusted at runtime. The amplitude, phase and offset for each axis can be adjusted independently.

3.4 Motion Parameters

The "Edit Motion Parameters" form, Fig. 6 allows the motion detection characteristics to be adjusted. The system can be set to detect motion towards the scanner, motion away from the scanner or either motion. "Threshold" sets the detection threshold used to discriminate object motion from noise. "Filter Constant" specifies the time constant used in the smoothing filter described earlier. Note that the scanner must be idling before these parameters can be changed.

3.5 Display Parameters

This form, Fig. 7 is used to adjust the scale of the display in the display box. Each axis can be adjusted independently. The entire display can be zoomed in or out.

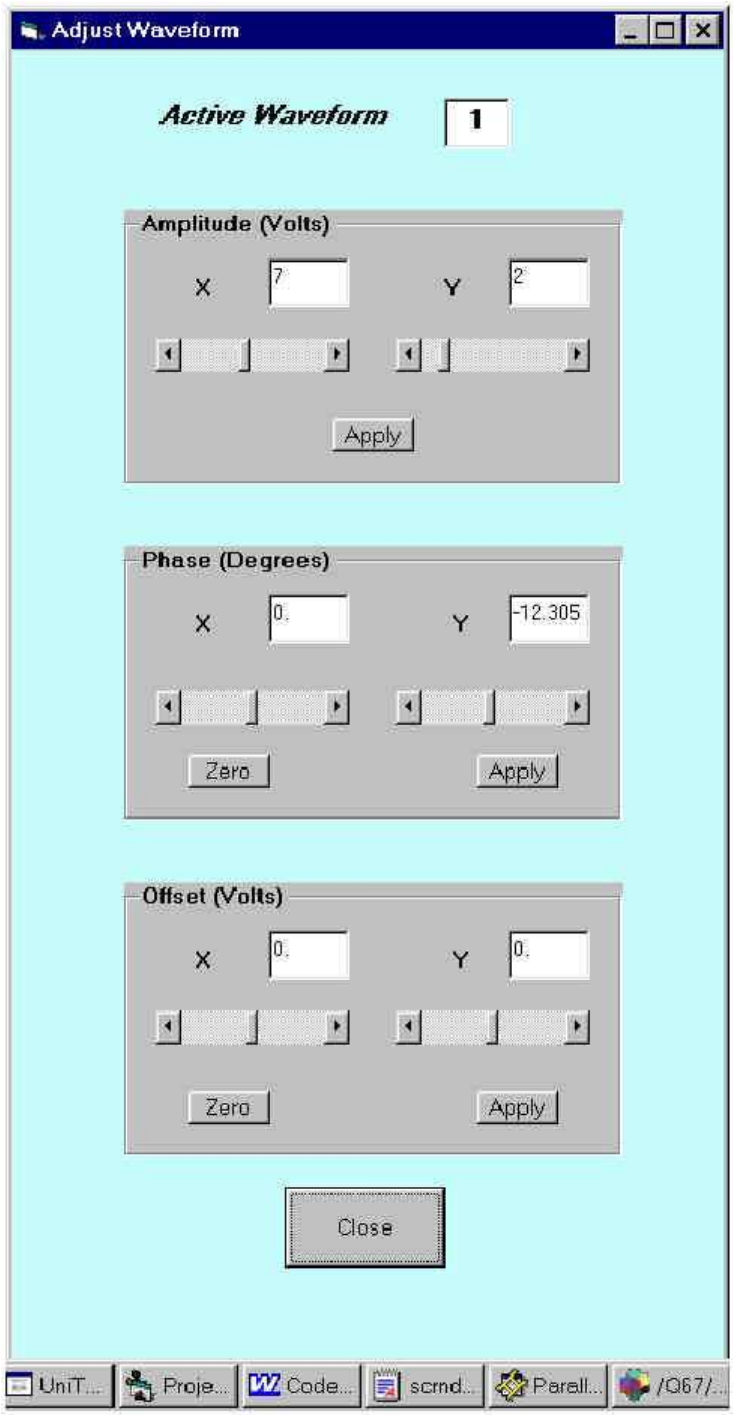

Figure 5. The Waveform Adjust Form.

This form supports dynamic waveform adjustment at runtime.

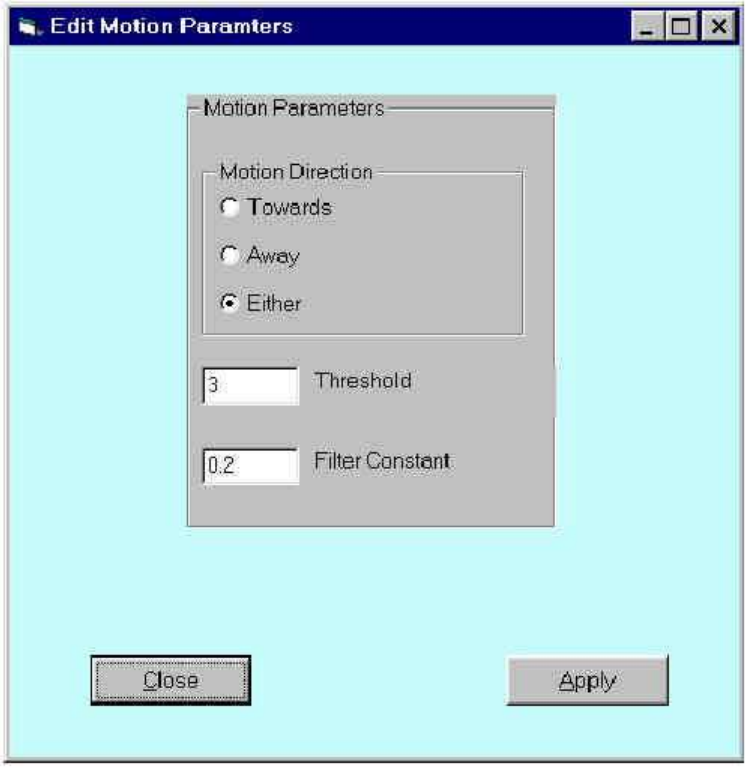

Figure 6. Motion Parameters Form.

This form supports the adjustment of motion detection parameters.

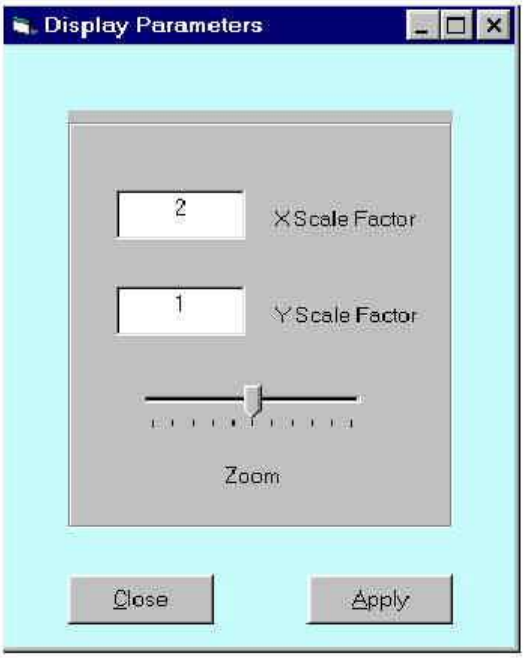

Figure 7. The Motion Display Form.

This form allows the display box parameters to be adjusted.

4. Hardware Configuration

The prototype laser scanner Fig. 8, is composed of two major components: the scanner head Fig. 9, and the system controller. The scanner head contains the CCD photodetector, the scanning mirrors, the galvos, and the lens. An external laser source is linked to the head via fibre-optics. Two galvo controllers are required for servoing the scanning mirrors. External custom electronics are used to acquire the raw CCD video signal. The scanner head is interfaced to the system controller through a custom designed scanner interface.

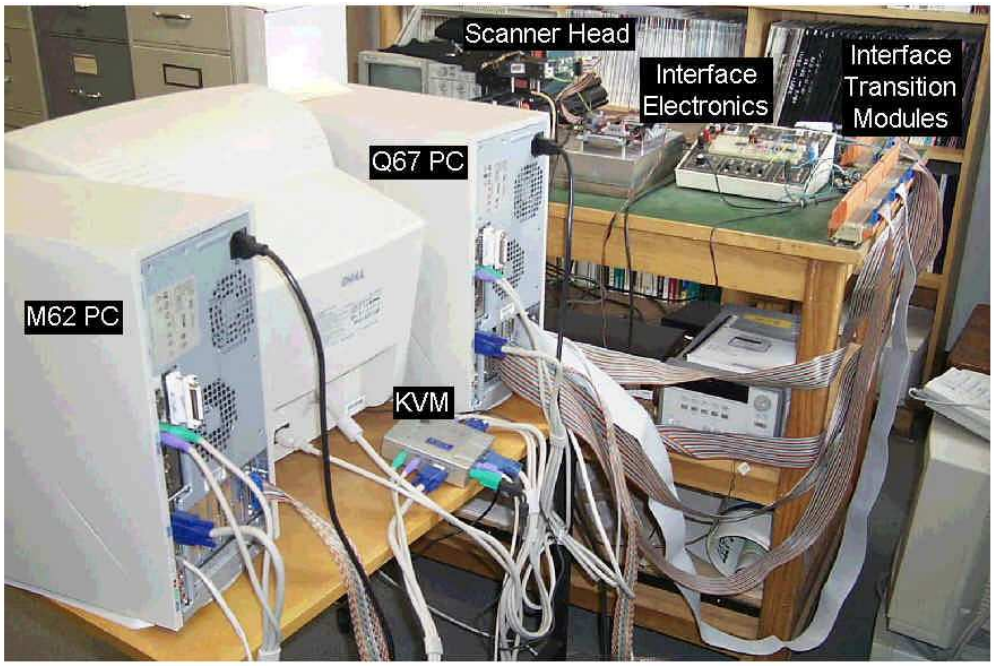

Figure 8. The Prototype Laser Scanner System.

The experimental system is composed of the scanner head, the interface electronics, the interconnection hardware, and the PC-embedded DSP system. The Q67 PC houses all DSP cards and serves as the runtime user interface and development environment for the Q67. The M62 PC is used only for M62 software development and debugging.

The system controller is based on a desktop PC platform which contains an array of embedded digital signal processors (DSPs), Fig. 10. known as the modular digital signal processor (MDSP)[7]. The PC supports the user interface, the mass storage and the network interface while MDSP supports the runtime operation of the laser scanner and the motion detection processing. The two components communicate via the PC's PCI bus. This section describes each component in some detail.

From a computing standpoint, the laser scanner Fig. 11, is composed of a dual axis galvanometer ("galvo") controller which generates the laser beam steering (using mirrors), and a position sensitive linear CCD photodetector which produces the raw range and intensity data. Scanning waveforms are generated by the MDSP system and are fed to the galvo controller while galvo positions can be read back from the controller.

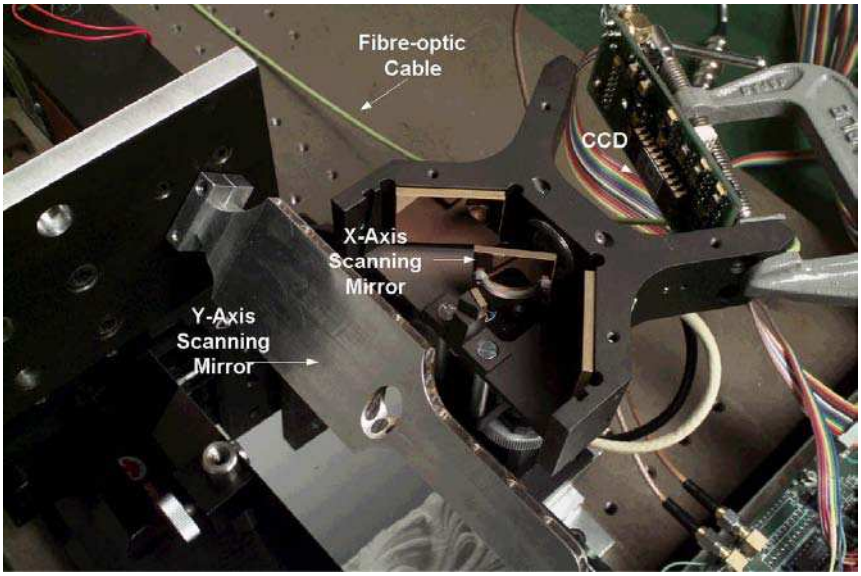

Figure 9. The Prototype Sensor Head.

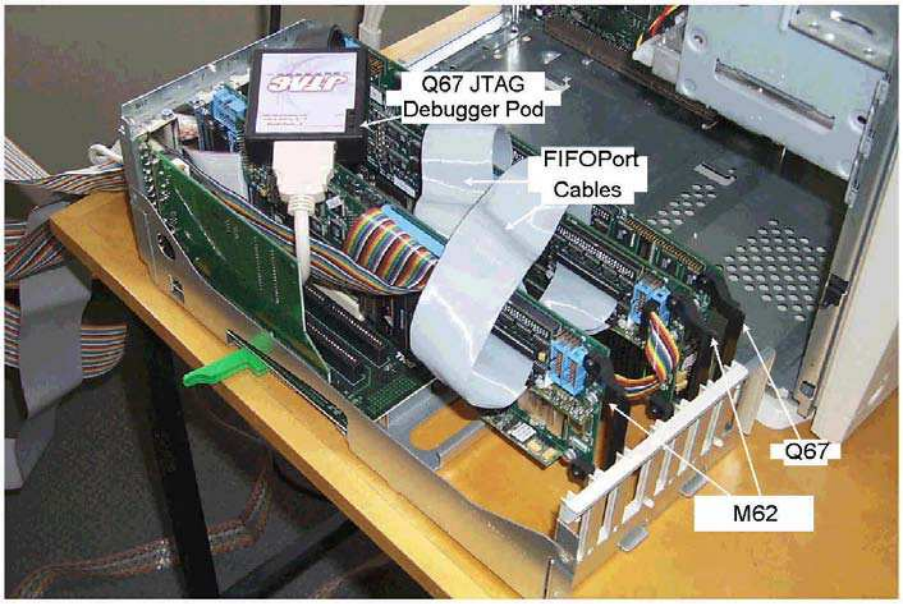

Figure 10. PC Cardcage Containing 3 DSP PCI Cards.

The M62 cards contain interfaces for the galvanometers and the CCD photodetector. The Q67 is the main DSP card. The FIFOPort interconnection cables that link the Q67 to the M62 cards are labelled. The JTAG debugger pod for the Q67 is shown but the pod for the M62 cards is not visible in the photo.

An analog filter and preamplifier preprocess data from the CCD photodetector. The data are then digitised by an A/D converter and passed to a DSP. With optical triangulation, range is directly related to the position of the laser peak on the linear photodetector. Therefore the accurate computation of the position of the laser peak is a critical element in the design of the laser scanner. A high precision peak detection algorithm[9] that yields sub-pixel resolution has been implemented on a DSP to compute the position of the peak in real-time. The low-level peak position and intensity data are transferred to another DSP where the motion detection algorithm is implemented.

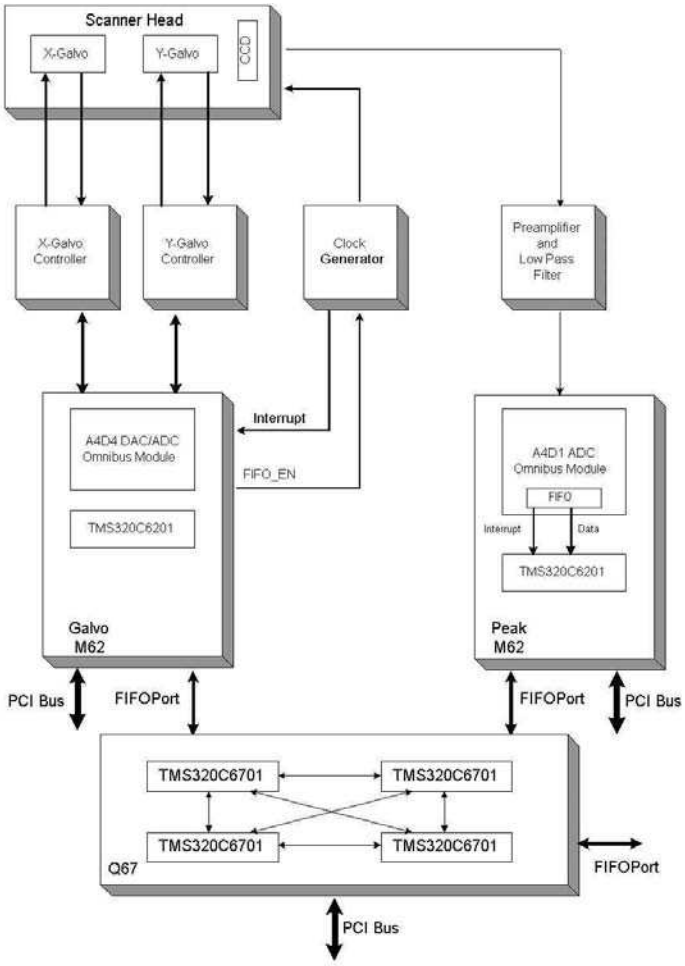

Figure 11. The Laser Scanner System Configuration.

Two low-level M62 DSP cards are used for waveform generation and for processing the raw range data. The main Q67 card handles high-level processing. All six DSPs communicate using dedicated high-speed bi-directional FIFO links. One Q67 FIFOPort remains uncommitted for system expansion. The three DSP cards reside on the host PC's PCI bus.

In normal operation, the laser scanner positions the laser spot and then measures the peak position and intensity of any laser light reflected back onto the CCD. By combining galvo position data with calibrated laser range data, the position of the target spot in camera co-ordinates can be determined. Calibration has not been implemented.

Section 4.1 describes the overall configuration of the system and the functionality of the desktop PC.

The modular DSP (MDSP) system will be described in Section 4.2. It is composed of a fully interconnected array of high performance DSPs that implement the laser scanner itself as well as the motion detection algorithm.

The prototype laser scanner will be described in Section 4.3. It is comprised of a modified single-axis scanning head with a second external scanning axis. Additional components include a laser light source, a galvanometer controller for each of the two scanning mirrors, and a power supply. External 'front end' electronics are used to preprocess the raw CCD data and to generate the clock and synchronisation signals for the system.

Interface electronics have been developed to manage the transfer of data and control signals between the DSP system and the scanner front end. This will be described in Section 4.4.

Correct operation of the scanner can only be achieved if the CCD photodetector and the scanning galvanometers are fully synchronised. Section 4.5 describes how this is achieved in hardware under the real-time control of a DSP.

4.1 The Desktop PC

The system is based on a desktop PC platform. The PC not only houses the DSP array (MDSP) but also provides the DSP development environment, the user interface, the file system, and the network interface. Physically, MDSP is comprised of three PCI bus DSP cards, which reside in the PCI backplane. Cabling between the scanner head and the MDSP system is interfaced via the PC backpanel (Fig. 8).

DSP software development required a second PC. This was due to the fact that two types of DSP cards were used in this system and each required its own development environment (see Section 6). A "KVM" switch was used to share a single development interface (keyboard, video monitor, mouse) with the two PCs. At runtime, only a single PC is needed.

4.2 The modular DSP (MDSP) system

MDSP is a scalable hardware/software system architecture based on multiple DSPs; it can readily accommodate the addition or removal of DSPs as the application demands.

Three different DSP cards are used in the current MDSP system, Fig. 10: a single "Quatro67" (Q67) card that contains 4 floating point TMS320C6701 DSPs clocked at 160 MHz, and two "M62" cards each containing a fixed point TMS320C6201 DSP.

The M62 cards were needed because the Q67 has no provision for analog or digital I/O. The M62 supports peripheral I/O using two mezzanine sites which can carry optional "Omnibus" interface modules[10].

Low-level processing of the raw CCD video signal is managed by one M62 clocked at 200 MHz. The raw CCD signal is digitised by an A4D1 Omnibus module and stored in a local FIFO buffer. The FIFO triggers an interrupt to the local DSP onboard the M62 when it becomes half full Fig. 11. This interrupt activates the peak detection process to be described later.

The two galvos are driven by analog waveforms generated by a second, 160 MHz M62 DSP card using DACs on an A4D4 analog I/O module. Raw galvo position data are digitised by ADCs on the A4D4. Galvanometer position updates are synchronised to the acquisition of range samples by using a hardware interrupt signal. The VCLK_INT interrupt is presented to the 'galvo' M62 card through an unpopulated Omnibus interface site (Site 1). It connects directly to an interrupt input, ExtInt2 available on JP23, pin 29 Fig. 12.

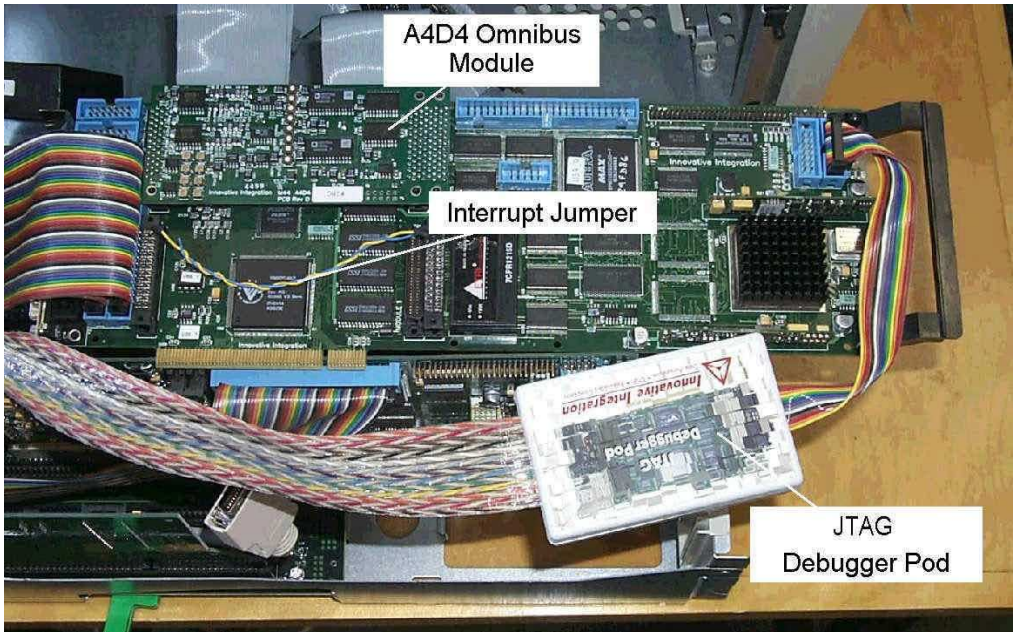

Figure 12. Configuring the VCLK_INT Interrupt.

The external VCLK_INT interrupt is used to synchronise the Galvo M62 DSP. It is hardwired through an unused Omnibus site as shown in the photo.

The Q67 is the main DSP system processor Fig. 11. All four onboard DSPs are interconnected via dedicated, high-speed, bi-directional, 32-bit data links ("FIFOLinks"). In addition, three off-board data links ("FIFOPorts") provide similar connections to the two M62 cards, however the FIFOPorts are limited to 16-bit transfers. One DSP on the Q67 is connected to the PCI bus to support communications with the host PC, but the other three are fully embedded. In this system, one FIFOPort remains uncommitted and is available for system expansion. Each M62 card has direct access to the PCI bus to support user interaction. This will be described in a later section that discusses system software.

4.3 The Prototype Laser Scanner

The prototype sensor head (Fig. 9) contains the electro-optical and electro-mechanical components. A fibre-optic cable carries the light from an external 10 mW laser source to the sensor head. The X and Y scanning mirrors are driven by General Scanning model G120DT and G325DT galvanometers respectively. These galvo driven mirrors provide the X-Y optical deflection for both the outgoing laser beam and the returning reflected light. A photodetector consisting of an EG&G Reticon RL0128TB 128-element CCD device is used for light capture.

4.4 The Scanner PC Interface Electronics

The prototype scanner head contains only electro-optics and electro-mechanics; no computer interface is provided. The external hardware that is used to interface the head to a computer is described in this section. This includes the clock generation logic, the camera control card and the synchronisation logic. The latter is used to synchronise raw range data acquisition (CCD video signal) with the laser pointing (galvanometer positioning). Hardware interconnections are also summarised in this section.

Clock Generation Card

The clock generation card, Fig. 13 produces both the CCD clock used to drive the CCD device and the "VOXEL CLOCK" which sets the voxel sample rate. The CCD clock must be allowed to run continuously to avoid saturating the CCD device. The CCD clock is derived from a 10 MHz crystal controlled oscillator, X1, not a 20 MHz oscillator as shown in the drawing. JP1 is set to position 1-2 to select a 5 MHz CCD clock output. U2 generates the VOXEL CLOCK. Jumper JP4 is installed to produce a VOXEL CLOCK after every 128 CCD clock pulses, since a 128-element CCD device is used. Both clock signals are output via MC3487 RS-422 line drivers.

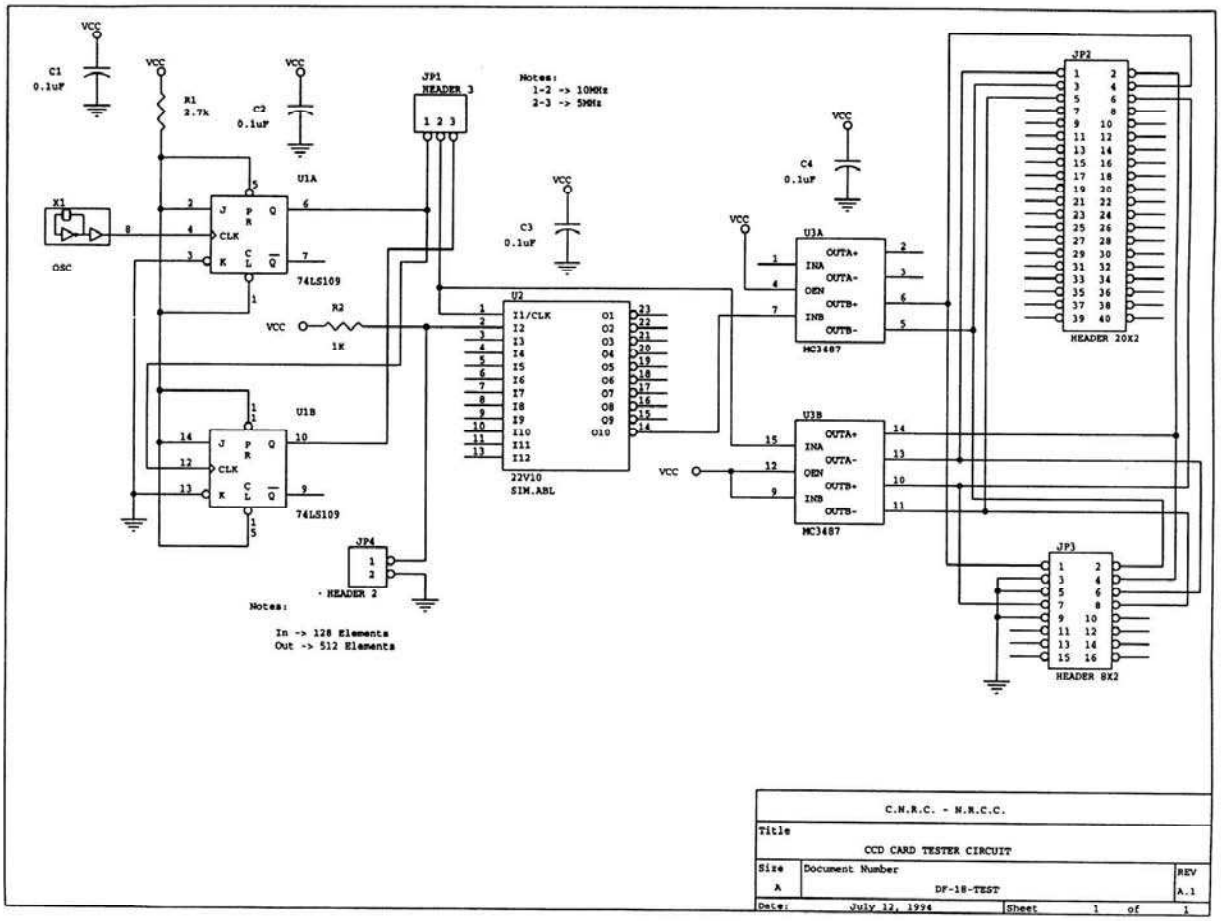

Figure 13. The Clock Generation Card.

A 10 MHz master oscillator (X1) is divided by U1A to derive a 5 MHz CCD clock (JP1-2). The VOXEL CLOCK signal (U2-14), which is derived from the CCD clock by U2, is used to set the sample rate of the scanner. Both signals are driven off-board using RS422 drivers (U3A/B) and are available from headers JP2 and JP3.

Camera Control Card

Fig. 14 shows an original drawing for the Camera Control Card. A number of modifications have been made for this project and will be described here. For our purposes, the main role of this card is to provide the initial analog preprocessing of the raw CCD video signal. However both the CCD clock and the VOXEL CLOCK from the

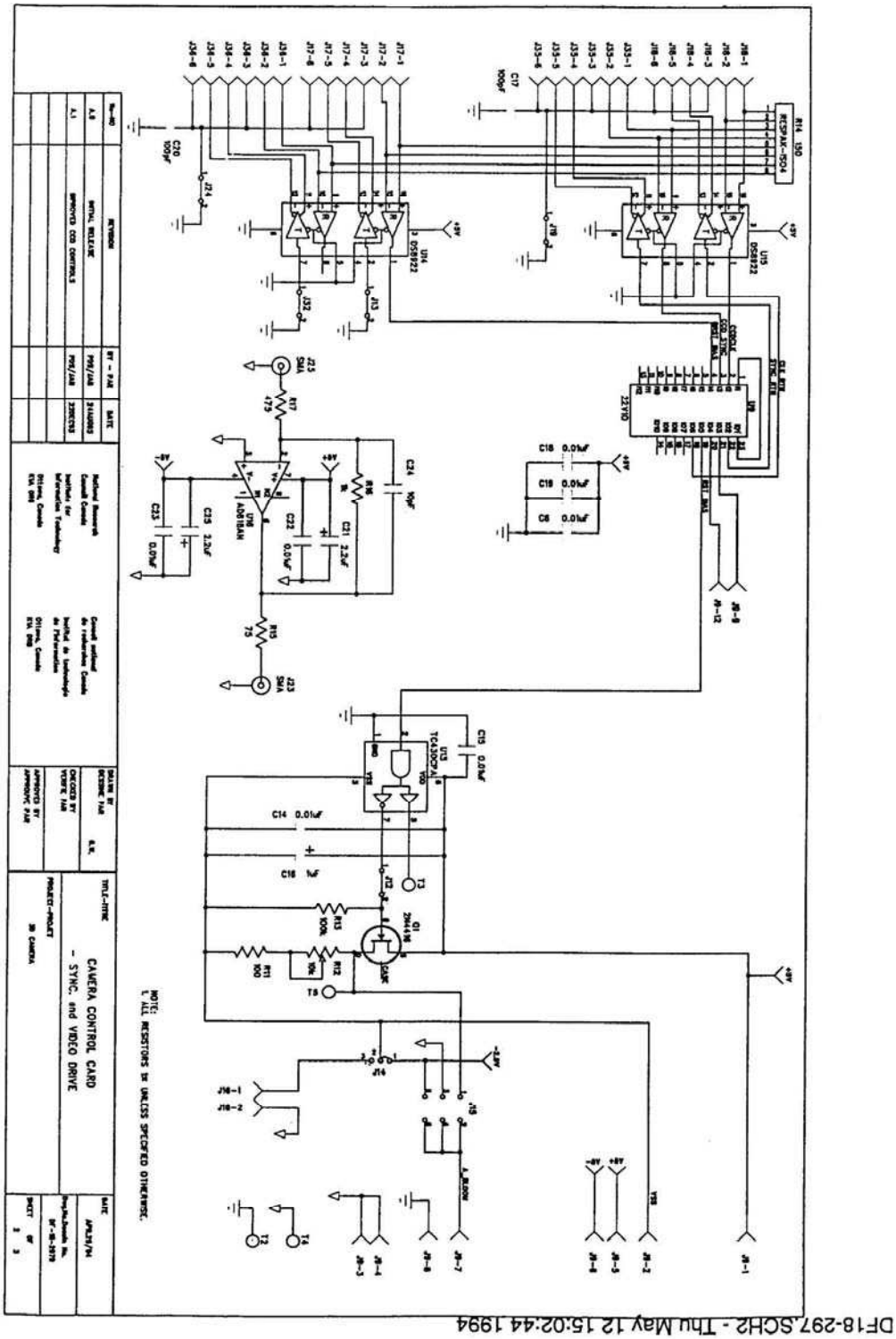

Figure 14. The Camera Control Card.

Raw clock signals are retransmitted by receiver/drivers (U15 and U14) on this card. The raw CCD "video" signal is filtered and amplified by U16. C24 and R16 determine the lowpass filter characteristics. The video signal from J23 is digitised by the ADC on the A4D1 Omnibus module, onboard the "Peak" M62 DSP card.

clock generation card are interfaced to this card via J18 and J35. The clock signals are handled by U15, DS8922 differential receivers/drivers.

The raw CCD video signal is presented to SMA connector J25. Here it is lowpass filtered and amplified by U16, an AD818AN op amp. The filter characteristics are determined by new values for C24 (82pf) and R16 (5.1k) and yield a lowpass cut-off frequency of about 380 kHz and a voltage gain of about 34 dB. R17 has been replaced by a 100-ohm resistor. The filtered video output is available from SMA connector, J23.

Commercial higher order LC lowpass filters can be inserted ahead of the inverting input to U16. Typically these filters require 50-ohm terminations and so will affect the gain of the preamplifier.

Synchronisation Logic

Fig. 15 shows the synchronisation logic which is used to provide exact control of CCD_CLK, the clock used for sampling of the CCD signal by the ADC on the A4D1 Omnibus Module. This clock is controlled by software to allow exact synchronisation with the galvo waveforms. The logic accepts as inputs the CCD clock, the VOXEL CLOCK and the software generated FIFO_EN signal from the galvo M62 DSP. It generates CCD_CLK, which clocks the A4D1 ADC and strobes raw data into the FIFO buffer. This logic also handles the free-running VCLK_INT signal, which is used to interrupt the galvo M62 DSP at the voxel sample rate. VCLK_INT interrupts the M62 galvo controller continuously even if the galvo controller is idling and waveforms are not being generated. DS8922 differential receivers/drivers, U1 are used to receive the input clocks, and SN75121 line drivers, U4 are used to drive the outputs.

When a command is issued to initiate a scan, the galvo M62 will asynchronously assert FIFO_EN and will then begin to step the galvos. FIFO_EN is used to enable CCD_CLK. However for the scanner to produce meaningful data, startup of the CCD_CLK must always coincide with the VOXEL CLOCK, i.e. both laser position control and CCD sampling must be synchronised Fig. 16. The synchronising logic delays the start of the CCD_CLK until the arrival of the next VOXEL CLOCK. CCD_CLK then remains synchronised for the duration of the scan.

U3 is used to synchronise the generation of CCD_CLK. Its output, U3-5 is asserted on the next occurrence of the VOXEL_CLOCK if FIFO_EN has also been asserted. It remains asserted until a VOXEL_CLOCK occurs in which FIFO_EN has been negated. This signal is used to gate the free-running CCD clock in U4B to produce the synchronised CCD_CLK output. This synchronisation guarantees that the ADC and the FIFO always capture a full complement of samples and that these samples are properly aligned with the galvo scans.

A number of discrete synchronising signals are also generated by the galvo M62 board using JP14, the Digital I/O connector. These signals include the FIFO_EN signal described earlier as well as the HYNC and VSYNC (horizontal/vertical synch) signals.

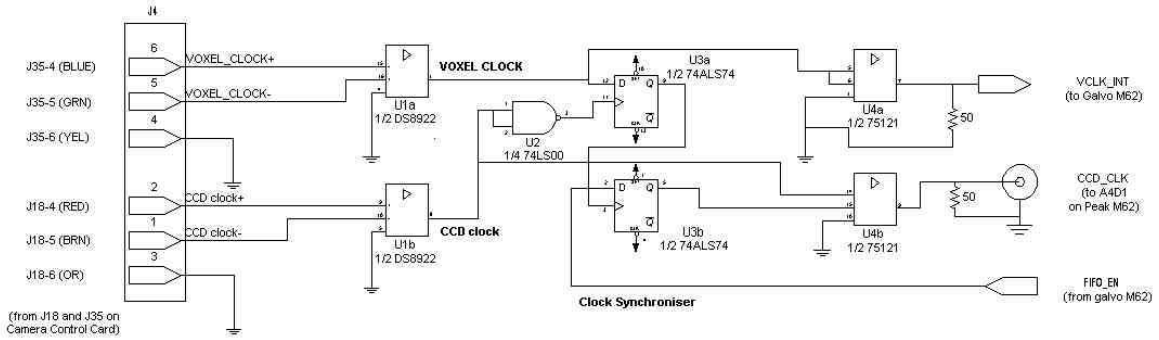

Figure 15. Synchronisation Logic.

This logic accepts the clock signals from the camera control card and the asynchronous FIFO_EN signal from the M62 DSP. It generates VCLK_INT, which by interrupting the galvo M62 sets the system sample rate, and CCD_CLK, which synchronously clocks raw CCD data into the ADC and FIFO.

Hardware Interconnections

Interconnections between the M62 DSP cards and the external logic consist of prototype cabling supplied by the DSP board manufacturer[10]. The M62 card uses industry standard 0.100" square double row 50-pin headers. Five 50-pin ribbon cables carry the signals to outboard transition modules (Weidmuller, RI 50A), Fig. 8. No attempt has been made to simplify this wiring. The five cables consist of three from the galvo M62 and two from the peak M62 card. The "Galvo cables" include: the synchronisation signals from the DIO interface via JP14, the VOXEL CLOCK interrupt signal on JP22, and the analog input/output signals from the A4D4 Omnibus module via JP18. The "Peak cables" include the CCD signals for the A4D1 Omnibus module on JP18, and the DIO signals from the DIO interface via JP14. A number of DIO signals on both the Galvo M62 and the Peak M62 have been allocated for test purposes. These allow timing measurements of peak detector and galvo controller performance.

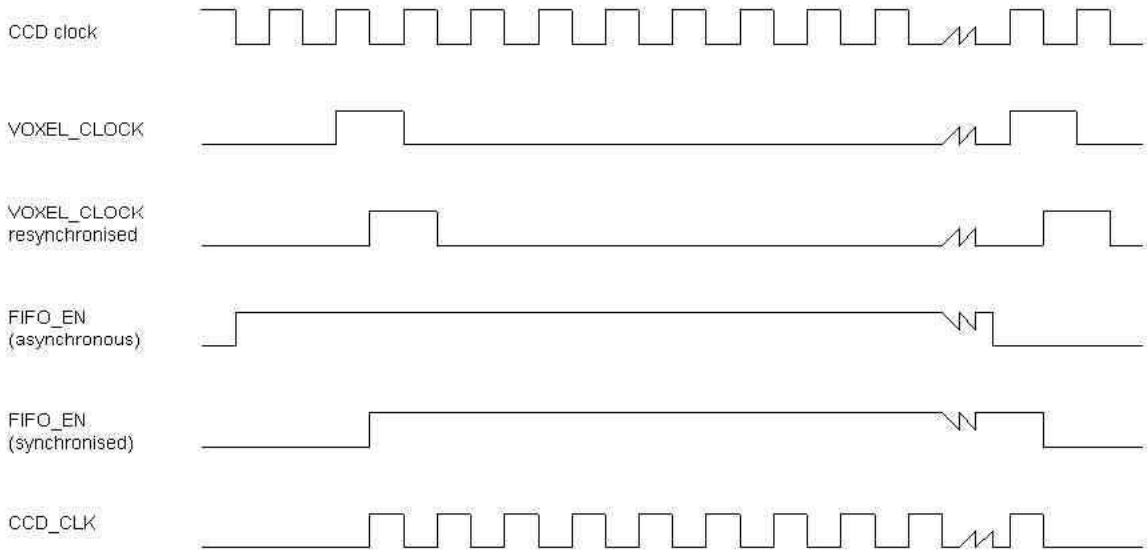

Figure 16. Synchroniser – Timing Diagram.

FIFO_EN is asserted asynchronously by the galvo M62 DSP. It is synchronised to the VOXEL_CLOCK and then used to gate the free-running CCD clock to produce the synchronised CCD_CLK. This guarantees that for each range sample, an exact number of raw CCD data samples are digitised by the ADC.

5. Software Configuration

3DMD has been implemented using an embedded array of DSPs based on three PCI bus circuit boards installed in a desktop PC. The DSPs are used to implement the 3D laser scanner and the motion detector while the PC is used for higher-level functions such as for data handling and as a user interface.

The DSP system has been divided into two major components: the two "front end" controllers, which manage the galvanometers and the CCD range data, and the main processor block, which performs higher-level processing such as for motion detection, and command and data handling.

There were two main design goals in the development of MDSP, the DSP architecture[7]: to produce a system that can meet the real-time constraints imposed by the 3D laser scanner, and to develop a scalable system that can be easily adjusted in the future to handle expanding requirements.

To meet these needs, we chose to implement the system software with a commercial real-time operating system (RTOS), MQX produced by Precise Software Technologies Inc.[8] This section describes the software structure that has been built. The design supports modularity through the use of multitasking and multiprocessing. Because of hardware

constraints, we had to use a message communication model rather than a shared memory model; message passing is used for intertask synchronisation and communication.

This section will begin with a brief outline of the main characteristics of the RTOS and will then present an overview of the system software. We will then describe the front-end DSP components that operate outside the RTOS abstraction and then discuss in some detail the RTOS task structure used to implement the high-level DSP system. We will follow this with short summaries of the messages that are used, and of the task creation sequence used at system startup. We will conclude this section with a description of the PC-based user interface, which has been written in Visual Basic.

5.1 General Features of the MQX Real-time Operating System

A multitasking RTOS facilitates the implementation of a complex system by allowing the software functionality to be partitioned and expressed as individual tasks. The RTOS provides mechanisms that support communications and synchronisation between the tasks. The additional ability to extend this multitasking abstraction across multiple processors adds considerable flexibility since this not only allows true parallelism to be achieved but also permits the number of processors in the system to be adjusted as the need arises. This is the approach used for 3DMD.

For this project we selected MQX from Precise Software Technologies Inc. We made a number of fundamental design choices at the beginning of the project. For example, we chose to use pre-emptive task scheduling rather than round-robin time slicing, and we chose message passing over semaphores or mutexes for intertask synchronisation. We chose message passing (Feb. 2001) because it was the only communications mechanism provided by MQX for use in multiprocessor environments. It is a more general solution, allowing us if necessary to easily reconfigure the software by moving tasks from processor to processor without limitation.

In MQX, tasks are assigned static priorities at compile time and may be created or destroyed dynamically at runtime. Task queues are used extensively by MQX. The “ready queue” maintains a prioritised list of tasks (first in, first out at each task priority level) which determines which task will be dispatched on the next pre-emption. In MQX, a task can be either running (the “Active” task), waiting to run but queued on the ready queue, or “blocked” (queued on some other queue). Tasks are added to or removed from the ready queue by the operating system kernel in response to events (creation of a task, arrival of a message, etc.) or by explicit function calls.

Once running, tasks communicate by sending messages to “message queues” which are usually created at startup. Generally, a message queue is owned by the task that created it. When a message arrives in a queue, the owner task, if it was awaiting a message, will typically be made ready to run. This is achieved by pre-empting the currently executing

task and adjusting the task "ready queue" for that processor. At the conclusion of this pre-emption, the highest priority task that is ready to run will be dispatched.

A number of system calls are provided in MQX to implement message passing. One of these, _msgq_receive() uses "blocking semantics" whereby the calling task will "block", i.e. be suspended from execution, until the arrival of a message. Indefinite blocking can be avoided by using a timeout. Other system calls for example allow a message to be sent to a destination queue or queues, or allow a task to poll a queue for a message without blocking. These calls allow considerable flexibility in configuring an application.

Some overhead is required in preparation to sending a message. For example, a message pool consisting essentially of blank messages must be created by the task before any messages are to be sent. Before sending a message, the source task must first fetch a blank message from the message pool and then fill in the necessary fields (source and destination addresses, size, data etc.). Finally, after receiving the message, the destination task must deallocate the message and return it to the message pool to avoid a memory leak. Message contents are user defined.

Interprocessor communications (IPC) between tasks on different processors is more complex and more expensive than for the single processor case. For MDSP, the low-level communications for IPC makes use of the FIFOLinks that interconnect all of the C6x DSPs on the Q67 processor board (Section 4.3). Because MQX had not been ported to this hardware, it was necessary for us to fully develop the IPC mechanism for this system. IPC is fully transparent to the user; messages destined for tasks on remote processors and those destined for tasks on the local processor only differ in the destination field of the message.

MQX uses a routing mechanism for multiprocessor message passing. The user must construct a static routing table for each processor. Each entry in the table corresponds to a mapping between the intended destination processor and a local queue which will trigger the IPC mechanism. For example, all messages intended for message queues on a remote processor "A" will be sent to a local message queue, for example "_id_A_q". This will trigger the IPC mechanism that will result in the message being transferred to the destination queue on processor "A". In an arbitrarily large system where processors are not fully interconnected, the routing table allows the user to define message routing that may involve hops between multiple processors before the message is finally delivered to its destination. In comparison to local message passing, there is a performance penalty associated with message passing to remote processors. The cost of this penalty is directly related to the complexity of the message route. Software design for any multiprocessor real-time system must pay careful attention to the fixed cost of message passing, and design the task configuration accordingly. Task to task intercommunication should be a deciding factor in determining task proximity.

For real-time systems, efficient interrupt handling is essential. To address this, MQX provides an interrupt handling mechanism called a "notifier". A notifier can be viewed as a high-level interrupt service routine (ISR) which possesses some characteristics of MQX

tasks. Typically, a notifier is installed very much like an interrupt service routine and is activated by a first level interrupt handler whenever that interrupt is asserted. Once active, the notifier usually clears the interrupt and fetches any volatile data. An important feature of a notifier is that it can also call any non-blocking MQX function. For example, it can send a message to a task that is currently blocked waiting to respond to this interrupt. This provides a simple and efficient way to synchronise task behaviour with external events.

We have measured the typical latency between the arrival of an interrupt and execution of an interrupt service routine on a 160 MHz C6201 to be about 1.0 µs. For 3DMD, notifiers are used on the Q67 for handling incoming data on FIFOPorts from the peak and galvo DSPs. These notifiers each trigger a task. The latency for the dispatch of the tasks cannot be measured directly however we estimate that it would be about 5 to 10 µs.

5.2 Overview of the System Software

The software for 3DMD consists of the PC-based user interface, and the real-time MDSP software, most of which runs on MQX. The PC-based user interface has been implemented with Visual Basic 6.0 and provides a number of forms to support user interaction (see Section 3).

Communication between the PC-based user interface and the embedded target DSP is based on an ActiveX control called DSPComponent[11], supplied by Innovative Integration, the manufacturer of the DSP boards. DSPComponent can be used directly by Visual Basic applications. It does however require a complementary board support DLL from the board manufacturer. DSPComponent works in conjunction with a library of functions which run on the DSP target. Together, they support communications between the PC and the target using mailboxes and interrupts. This will be outlined in more detail in the next Section.

MDSP exploits the multitasking and multiprocessing capabilities of MQX. Early measurements of IPC performance played an important role in the design of the system software. Preliminary performance measurements indicated that message traffic between “Peak M62”, the DSP that handles the raw CCD video data (Fig. 11) and Q67, the main DSP board, would exceed the IPC capabilities. To avoid this limitation, we decided to remove Peak M62 from the RTOS abstraction, in effect defining it as a “custom peripheral”. The same decision was taken for the “Galvo DSP” which controls the scanning galvanometers. Both DSPs communicate with the Q67, the main DSP system, entirely through the bi-directional “FIFOPorts” using low-level communication functions. On the Q67, an MQX-based task structure was developed to manage the communications with these custom peripherals and to integrate them into the main body of the system. This will be described in some detail in Section 5.5.

5.3 Host PC and Embedded DSP Target Communications

DSPComponent was introduced in Section 5.2. It provides an ActiveX component, which can be integrated directly into a Visual Basic application, and which supports runtime communications between the Visual Basic program and the target DSP system. DSPComponent includes a number of properties, methods and events. A set of functions must run on the target DSP to complement this functionality. The board manufacturer provides these functions in the form of a runtime library.

DSPComponent includes a bi-directional interrupt mechanism. A target DSP can call mailbox_interrupt() to trigger an interrupt on the host PC. The incoming interrupt at the PC will trigger an "OnInterrupt" event that automatically activates a user-written event handler object. The event handler will use the "InterruptAcknowledge" method to handle and clear the interrupt and read any associated data. Similar methods exist to transfer interrupts in the opposite direction.

DSPComponent also uses mailboxes for exchanging data between the host PC and the target DSP. The target DSP calls write_mailbox() and read_mailbox() to transfer data to or from a mailbox while the host PC uses the "Mailbox" property for the same purpose. Other properties or functions are provided to allow the host or target to check the status of a mailbox for polling purposes.

5.4 The Peak Detector and Galvo Control DSPs

For efficiency, DSPs have been dedicated to two critical front-end processes: CCD data handling, and galvanometer control. These DSPs are treated as custom peripherals that lie outside the multitasking abstraction. The Peak DSP handles the raw CCD data and extracts raw range and intensity data corresponding to each sample point. These data are fed continuously to the main DSP system. The Galvo DSP manages waveform generation and returns galvo position data to the main DSP system for further processing.

While the system is actively scanning, analog CCD data are generated continuously. These data are digitised by an ADC onboard the A4D1 Omnibus module on the Peak M62 and are stored in a FIFO buffer, Fig. 11. When the FIFO becomes half full, an interrupt is triggered to the Peak M62 DSP. The DSP interrupt service routine transfers these data using direct memory access (DMA) to local DSP memory. It then executes a peak detection algorithm[9] which computes not only the position of the peak with sub-pixel resolution but also the corresponding intensity of the peak. The peak position represents the CCD element actually illuminated by the returned laser spot and is related through triangulation to the range to the target. Range and intensity data are then transferred to the Q67 via a FIFOPort.

Both front-end DSPs are heavily loaded. We have measured the 200 MHz Peak M62 DSP load to be about 69% when the CCD is clocked at 5 MHz. This does not allow sufficient time to implement message passing to tasks on the main DSP board. For this reason, this software runs independently and simply passes the data to the main DSP via a FIFOPort. The peak DSP is a free-running system, it requires no input commands to operate.

The 160 MHz Galvo DSP is integrated into the system in a similar manner. For the 3DMD application, it is running at about 97% capacity. Unlike the Peak DSP, the Galvo DSP fetches incoming commands to handle initialisation and runtime waveform adjustments. The Galvo DSP also returns position data for the two galvos to the main DSP using a FIFOPort.

A FIFOPort buffers data received by the main DSP, either from the Peak DSP or the Galvo DSP. When a FIFOPort buffer becomes half full, an interrupt is generated to the destination Q67 DSP which then transfers the incoming data. This is done within the multitasking abstraction using notifiers.

5.5 3DMD System Task Configuration

The application software, which runs under MQX, has been partitioned into tasks, Fig. 17. Initialisation and task creation will be described in Section 5.7.

Idle State

After initialisation and the creation of all tasks, the system enters an ‘idle’ state awaiting input commands from the user interface. The Galvo DSP continuously responds to and discards incoming VOXEL_CLOCK interrupts since no scan is active. The Peak DSP simply idles.

Parameter Setup

From the idle state, the user has complete access to all system parameters. The user can define and adjust the scanning waveforms, change the motion detection parameters, and start and stop a scan.

User requests to define or adjust a scanning waveform are handled by the User_Interface_Task() on DSP_A. The PC-based graphical user interface formats all such requests into commands which are sent directly to the DSP system via the mailbox mechanism. The User_Interface_Task() polls the mailbox for incoming commands. In response to each request, it will generate a message to the Galvo_Control_Task() on DSP_B, which handles direct communications with the Galvo DSP.

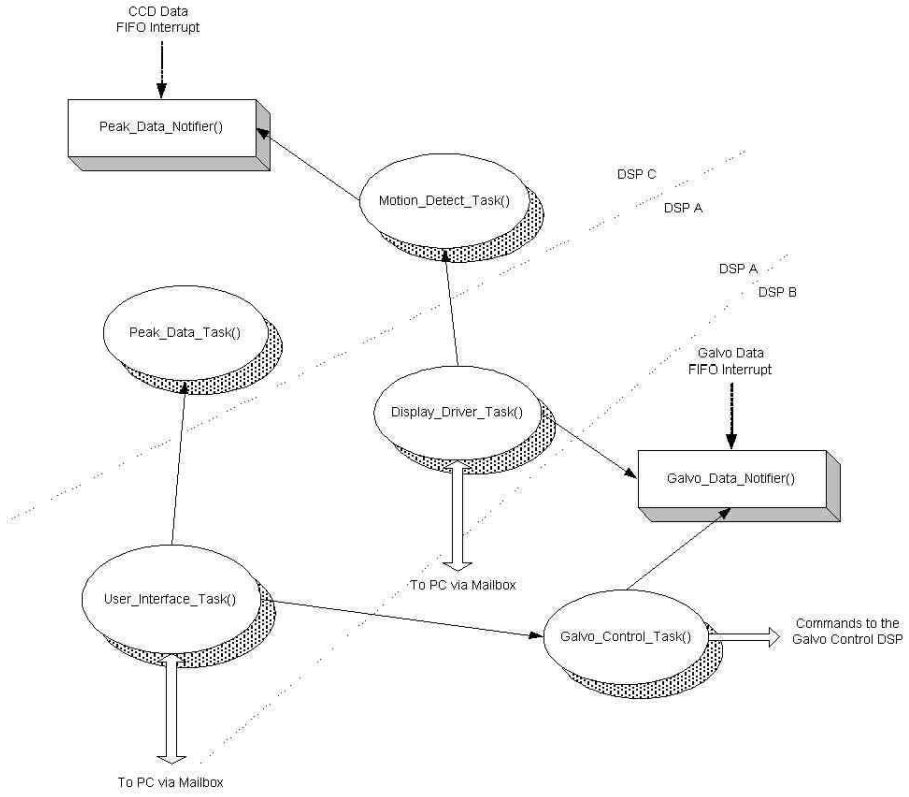

Figure 17. The Task and Message Structure for the MQX-based 'Q67' Multiprocessor DSP System.

Five tasks distributed across three DSPs are used in the implementation of the 3DMD prototype. Two additional dedicated DSPs provide low-level control and processing for CCD data acquisition and galvanometer control. These 'external' DSPs feed data to the main board using 'FIFOPort' data communication links where interrupt triggered 'notifiers' handle the incoming data. The User_Interface_Task() and the Display_Driver_Task() communicate with the PC-based user interface using the PCI bus on the PC's backplane.

The Galvo_Control_Task() is normally blocked, waiting to receive a message from the User_Interface_Task(). An incoming message will unblock the task which will then proceed to interpret the command. The Galvo_Control_Task() will then generate a low-level command directly to the Galvo DSP and after the transfer is complete, will send an acknowledgement message to the User_Interface_Task() to complete the message handshake. On receipt of the acknowledgement message, the User_Interface_Task() resumes polling for the next incoming user command.

The repertoire of commands for the galvo control DSP include: initialise galvo, start galvo, stop galvo, define waveform, change amplitude of current waveform, change phase of current waveform, change offset of current waveform and select another waveform. A total of 16 waveforms can be defined and stored by the Galvo DSP. Waveforms can be changed dynamically at runtime by simply selecting its wave index.

Active Scanning

In response to a user request to start a scan, the Peak_Data_Task() initialises its internal data structures and resynchronises its communications with the Peak DSP. Also, the Galvo_Control_Task() issues a START_GALVO command message to the Galvo DSP. This command is interpreted by the DSP which then begins to generate the two scanning waveforms. In addition, the Galvo DSP enables the generation of CCD range and intensity data by asserting the FIFO_EN signal to the synchronisation logic (Section 4.4 and Fig. 11). The range and intensity data are now transferred to the FIFOPort that connects the Peak DSP to the main Q67 DSP board.

As the Q67 FIFOPort fills, it generates interrupts that are handled by the Peak_Data_Notifier(). With each interrupt the Peak_Data_Notifier() uses a DMA transfer to move the contents of the FIFOPort to a local buffer that is managed by the Peak_Data_Task(). This local buffer contains the raw range and intensity data that are used by the motion detector.

Motion Detector

The motion detector not only processes the range data for the detection of objects in motion but also prepares them for transfer to the PC for real-time display. The Motion_Detect_Task() is implemented as a loop. At the top, the task sends a message to the Peak_Data_Notifier(), requesting the latest complete line of range data. In its loop, the notifier polls for an incoming message from the Motion_Detect_Task(). (By definition, a notifier cannot call blocking functions, since this is inconsistent with proper design of interrupt handlers.) If a request message is found, the notifier responds by sending a message containing a 'structure pointer' to the Motion_Detect_Task(). The structure contains the information which allows the Motion_Detect_Task() to directly access the latest acquired line of range data.

The motion detection algorithm (Section 2.2) is then executed. The 'index' (position along a scan line) and 'value' (departure from average position) of any points that surpass some threshold, are stored in a message buffer that can be sent to the Display_Driver_Task(). The Motion_Detect_Task() will poll for a message from the Display_Driver_Task(). If a request message is available, then the message is sent and execution returns to the top of the loop. Note that if the Display_Driver_Task() has not reported with a new message, the Motion_Detect_Task() continues without interruption. This means that motion detection is continuous, and independent of the performance of the display system.

Display Driver

This task handles the transfer of motion data from the embedded DSP to the user interface display that is built using a Visual Basic 'picture box' on the PC. The

Display_Driver_Task() requests data from both the Motion_Detect_Task() and the from the Galvo_Data_Notifier(). It receives all of these data and combines them into a single data structure, which contains points in motion, their intensity (value) and their positions (X- and Y-galvanometer positions). These data are all sent to the display application on the PC.

The galvo position data are derived directly from the galvo controller's position sensors. Ideally, the waveform displayed on the user interface should be identical to the actual laser scan seen on the target. However, we have found that the displayed waveform is slightly distorted (see Section 2.3).

An alternate implementation had been used earlier in which the galvo positions were computed based on the index of the sample points. Since the waveforms are predetermined, it is easy to compute the corresponding X-Y co-ordinates for each point. However, this method does require careful calibration. For any set of scan parameters, a fixed delay will exist between the *generated* galvo waveform drive voltages and the *actual* galvo positions. These phase delays must be measured carefully and included in the calculation of the final position of each sample.

Comment

It should be clear from the above that the use of multitasking with message passing yields a very flexible architecture. In general, communications between software objects (tasks) can be easily configured regardless of which processors are actually involved. Of course, careful placement of tasks is required to meet performance needs. For example, the assignment of the Motion_Detect_Task() and the Peak_Data_Task() to the same processor was done to give the motion detection algorithm direct access to the raw range data. This avoids unnecessary and expensive interprocessor data transfers.

5.6 Summary of System-wide Message Passing

User defined messages are used throughout this application. Each message type is defined as a dedicated data structure. For example, the message type used for messages sent by the User_Interface_Task() to the Galvo_Control_Task() are of type GALVO_CMD_MESSAGE_STRUCT.

```
typedef volatile struct galvo_cmd_message
{
    MESSAGE_HEADER_STRUCT     HEADER;
    uint_16                   GALVO_CMD;
    uint_16                   WAVE_INDEX;
    WAVEFORM_DEFINE_STRUCT    WAVEFORM_DEF;
} GALVO_CMD_MESSAGE_STRUCT, * GALVO_CMD_MESSAGE_STRUCT_PTR;
```

All message definitions include a HEADER structure that defines the message size and the message source and destination queues. This message also includes a galvo command and a waveform index. Finally, the message contains a waveform definition structure that is used to pass new waveform parameters to the Galvo_Control_Task(). Details of the message types appear in the application software that is included in the appendix.

The various message types that are used in 3DMD are summarised in Fig. 18. Message names are declared locally in each task and are not consistent from task to task. Therefore, Fig. 19 uses a representative set of meaningful message names. It is interesting to note the clear distinction between control message traffic involving the User_Interface_Task(), the Peak_Data_Task() and the Galvo_Control_Task(), and the data message traffic that links the two notifiers with the Motion_Detect_Task() and the Display_Driver_Task. Note that this application does not make any use at all of a fourth DSP, DSP_D.

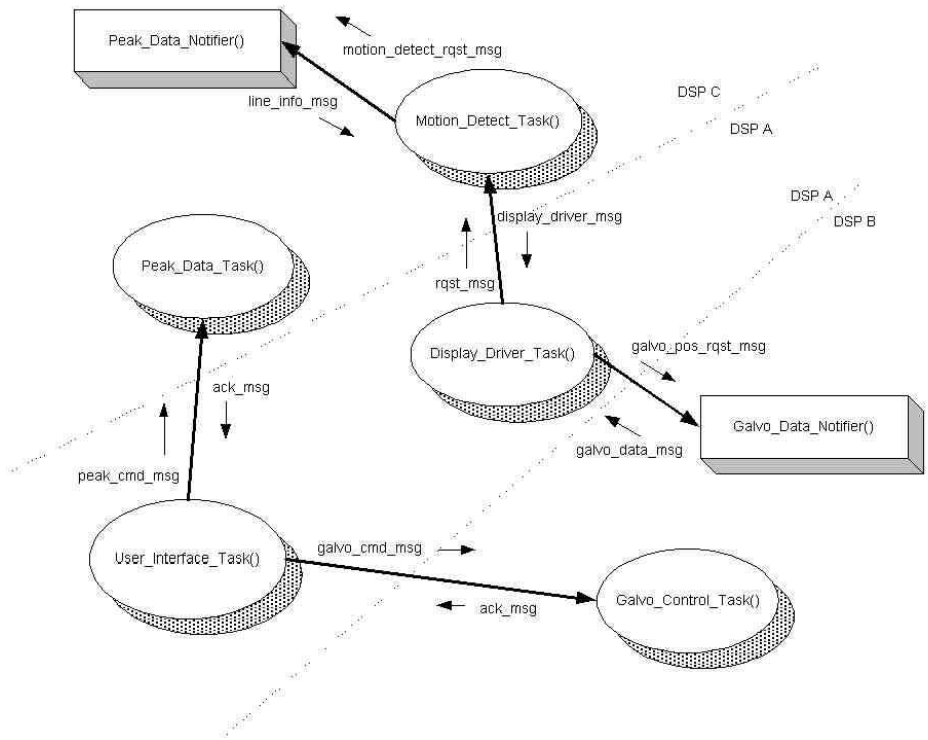

Figure 18. Summary of Message Passing for 3DMD.

Message exchanges between corresponding tasks relies on a message handshake. The large arrow interconnecting each pair of tasks represents the direction of the originating message. Typically, the originator blocks until a reply message is received. The small arrows indicate the direction of the individual messages.

5.7 Task Creation Sequence at Startup

In a multitasking, multiprocessor environment, system startup must be carefully managed. Tasks must be created in a predetermined sequence. Also, essential user declared resources, such as message pools and message queues that are required for user message passing, must also be created and initialised before normal system operation can commence.

Applications written under MQX make use of a "Task Template List" to define the characteristics of all tasks that may be created on each processor[12]. One of these characteristics is the 'auto-start' feature. On startup, after MQX has completed its internal initialisation, it will add to the ready queue any task that has been declared "auto-start". In this application, one user task is declared this way. In addition, it should be noted that interprocessor communications (IPC) is managed on each processor by a dedicated task called _ipc_task(). This task must be declared 'auto-start' on each processor to allow the multiprocessor system to initialise. Fig. 19 illustrates the task startup sequence.

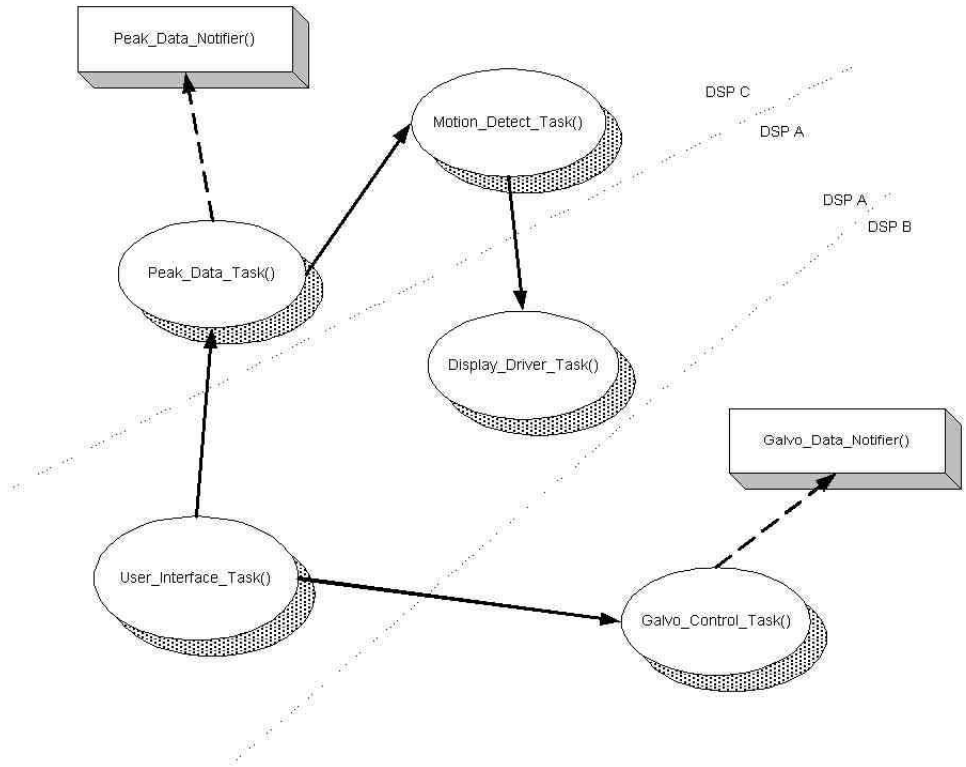

Figure 19. Startup Task Creation.

Task creation sequence at system startup is shown by solid arrows. The User_Interface_Task() is started automatically after MQX completes its initialisation (MQX_AUTO_START_TASK). Once started, this task creates the Peak_Data_Task() and then the Galvo_Control_Task(). The Peak_Data_Task() creates the Motion_Detect_Task() which then creates the Display_Driver_Task() on DSP_A. Finally, notifiers are installed (dashed lines) by the Peak_Data_Task() and the Galvo_Control_Task().

5.8 Communications Aspects of the User Interface Software

The functionality of the PC-based user interface was described in Section 3. It is written in Visual Basic 6.0 and uses DSPComponent to support runtime communications with the embedded DSP system. A code listing is included in the Appendix. This Section will briefly outline the design of this software, concentrating on the communications with the embedded DSP system.

"frmMain" begins to execute when the application is launched. It provides pull-down menus for system configuration, control frames with command buttons for initialising the system and for starting and stopping the scanner, and display boxes for target responses (error or status reports) and for presentation of detected motions.

While "frmMain" includes code to handle all of the command button click events, its most interesting element is "dsp_OnInterupt()", the event handler which is triggered by a mailbox interrupt generated by the arrival of data from the embedded DSP.

Incoming messages from the embedded DSP use a simple protocol which identifies the length of the incoming message and which distinguishes text messages (for status and error reports) from data messages (for graphical display).

Two user-level functions were written for the DSP target to support data transfers to the host PC. "mbox_display_data()" transfers an array of points to the host for display while "mbox_printf()" provides the same functionality as the printf() function in the C Standard Library except that the output is directed to the mailbox rather than stdio. Internally, each function follows a similar procedure. First an interrupt is sent to the host PC by calling the mailbox_interrupt(count) function. The "count" argument specifies the number of mailbox transfers that will follow in this transaction. Next the message type is sent using the low-level write_mailbox() function. This identifies the following data as either a text string or a set of graphical data. Finally the contents of the message are sent, in a series of transfers using write_mailbox().

In response to an incoming interrupt at the host PC, DSPComponent will activate the "OnInterrupt" event handler which is instantiated here by dsp_OnInterrupt(). dsp_OnInterrupt() parses the incoming message and extracts the "count" and "message type" information. It then proceeds to fetch the remaining data and present it either to the display box or to the picture box, depending on the message type (text string or graphical data). When presenting motion data, the dsp_OnInterrupt() handler displays range data position and intensity, and also displays a bounding box of computed mean position and standard deviation. The performance of the host PC and the mailbox communications mechanisms determine the display update rate. Motion data are always generated in real-time regardless of the performance of display handling.

6. The MDSP Development Environment

This section discusses software development using MQX, and runtime DSP debugging using Code Composer Studio (CCS)[13].

6.1. Software Development with MQX

Installation of MQX

The installation procedure for Precise MQX is described in the Release Notes that are packaged with the product. ARC International who now produces Precise/MQX has divided the system into a Processor Support Package (PSP) and a Board Support Package (BSP). A PSP deals only with processor specific issues while the BSP handles details that are board specific. In our case, the PSP for the TMS320C6201 and the floating point TMS320C6701 are essentially identical. Only a demonstration BSP was provided with the system, BSPs for the Q67 and M62 cards had to be developed by the author.

The ultimate goal in the preparation of the PSP and BSP is to generate runtime object libraries. Over the course of this project, many changes have had to be made to both packages and the libraries have needed to be rebuilt. Most of these changes are documented in the Appendix which describes the detailed procedure to install MQX and to build working BSPs for these targets.

Support for interprocessor communications (IPC) for our hardware was not included with the original PSP/BSP package. That development, which was undertaken here, was complex and is beyond the scope of this report[14]. During that process, changes were made to the 'build' process to accommodate the new IPC code and the new files which were added to the directory structure of MQX.

Special Aspects of Software Design using MQX

Our DSP system can be classified as a loosely coupled multiprocessor system. Each processor has its own private memory; there is no shared memory. Furthermore, processors are interconnected by dedicated high-speed bi-directional FIFO buffers. This configuration of processors requires that each DSP maintain its own copy of the RTOS kernel.

During software development, code must be developed separately for each processor. MQX uses the concept of a "Task Template" to describe the characteristics of each task that may eventually run on a processor. This is described in detail in the MQX User's Guide[12].

IPC provides the infrastructure for intertask communications across multiple processors. In a multiprocessor application, MQX requires the user to build a routing table for each processor to handle IPC. Furthermore, an initialisation table must be constructed on each processor to configure the functions and queues used for IPC. See the MQX User's Guide and the Appendix for more details.

6.2. Debugging a Multiprocessor System with Code Composer Studio

The Hardware Debugger

"Code Hammer" JTAG hardware debuggers, supplied by Innovative Integration were used for debugging[10]. JTAG debuggers work using a serial scan path to move data serially into and out of the digital system.

Two debuggers were required: one for the 4 DSPs on the Q67 board and a second that was shared between the 2 M62 cards. It was not possible to link the scan paths of the 3 boards and use a single debugger for the entire system. Since only one debugger can be installed in a single PC, it was necessary to use 2 PCs for system development. We allocated one PC for the Q67 development and a second for the M62s. This also simplified the management of the development environment since the 2 board types also require different runtime libraries and different system configuration options (default directories etc.).

All DSP boards physically reside in the Q67 PC chassis. The second PC houses the debugger and the development tools for the M62s. The M62 debugger is cabled to either one of the M62 cards on the Q67 chassis (Fig. 10). Moving the debugger from one M62 to the other requires turning off the power to the Q67 chassis, and physically moving the debugger cable from the JP11 connector on one card to the same connector on the other.

Outline of Code Composer Studio

Code Composer Studio (CCS) is supplied as part of the code generation toolset by TI, who manufactures the TMS320C6x family of DSPs[13].

CCS inserts a high-level graphical user interface on top of the hardware debugger. Some of the major debugging functions supported by CCS include executable image download, runtime control (starting, pausing, single stepping (assembly-level or C source-level), halting), breakpoint planting, editing of memory and registers, call stack display, graphical data display and an stdio-style output window. The output window is called a general extension language (GEL) output window and is very useful in the development of embedded systems that provide no user I/O display windows of their own. CCS supports parallel debugging of multiprocessor systems through startup and shutdown synchronisation and by presenting a dedicated display window for each processor.

JTAG debuggers do not provide the real-time trace capability normally found on much more expensive in-circuit emulators. Furthermore their response time tends to be much slower due to the serial scan path.

We were forced to use CCS for downloading the DSP targets. The board manufacturer supplies, as part of the development toolset, a utility for loading small application programs directly to on-chip memory. Unfortunately application code cannot be downloaded to off-chip memory using this utility[15]. Since on-chip memory is limited to 64 kB, our MQX applications cannot be directly loaded and executed using this utility. CCS and an external debugger are required for that purpose.

6.3. System Startup

Before running a multiprocessor application, executable images must first be downloaded to each processor. System startup must be carefully synchronised to avoid race conditions since CCS starts each processor in sequence via the JTAG debugger. Section 5.7 discusses issues concerning the startup of user-defined tasks. Startup delay is more complicated when a remote, PC-based user interface is used. In this case, the Visual Basic interface software must also be designed to synchronise with the DSP system at startup. We use a simple handshake for that purpose.

DSP startup includes some short initialisation messages from the embedded DSPs. These messages are directed to the general extension language (GEL) output window. Since this mechanism uses the JTAG scan path, the output is very slow. The Visual Basic application must allow the startup to complete before proceeding. Failure to do so results in a system stall.

6.4. Software Development Tools

Application software is written in C and is built using the TI Code Generation Tools[13]. A number of independent libraries are required at linkage editing time when DSP applications are developed using MQX. These libraries include the TI C runtime libraries, the DSP Peripheral Library (available for each DSP board from Innovative Integration's "Zuma Toolsets"), and the MQX PSP and BSP libraries.

Attempts to upgrade system components often come with unexpected side effects. For this project we continued to use CCS Version 1.0 even after newer versions became available. We discovered that changes to some GEL functions in newer versions of CCS were incompatible with our version of MQX. We also found that newer releases of DSP Peripheral libraries would not boot. In the end we used the following tools and versions:

- Precise/MQX Version 2.40
- Code Composer Studio Version 1.0
- TI TMS320C6x C compiler Vers. 3.01
- Innovative Integration Q67 Zuma Toolset Vers. 1.06
- Innovative Integration M62 Zuma Toolset Vers. 1.16

6.5. Managing Complexity of MDSP Software Development

A variety of components are used in the implementation of 3DMD. These include libraries from three sources, two different hardware platforms, and code generation tools. Over the course of a project such as this, it can be expected that new releases of software and of software tools may become available. In the future, new hardware platforms themselves may also be incorporated. We have developed a directory structure and a development strategy that manages the complexity associated with an evolving system such as this. We will first describe the directory structure and then the process to generate executable images.

Directory Structure

A tree-structured directory is used for 3DMD and all other demonstration programs. Branches in the tree are used to identify variations in hardware or software components that may arise in developing different demonstration versions. Different branches may exist representing different "Zuma" toolsets or different versions of MQX. The following is an example of a Q67 directory tree rooted in the "Working" folder.

Working | C6xScanner | MQX2.40 | MotionDemosB | demo0 | iiq67 | inc | z106, and
Working | C6xScanner | MQX2.40 | MotionDemosB | VB.

Directory	Description
C6xScanner	Main branch
MQX2.40	RTOS version 2.40
MotionDemosB	Program family name
demo0	This demonstration program name
iiq67	Target board type (Q67 DSP), contains source files
inc	"Include" path
z106	directory for intermediate and binary files
VB	Visual basic companion application

Changing to Zuma toolset release 107 would result in a parallel tree:
Working | C6xScanner | MQX2.40 | MotionDemosB | demo0 | iiq67 | inc | z107 since no changes would be required in the source code. A similar tree structure is used for the M62 DSP applications with the exception that MQX is not used in 3DMD.

Generating Executables

Runtime executable images are produced using the "Makefile" utility in CodeWright. The utility is included in the "Zuma Toolset" provided by Innovative Integration[16]. Each executable image is developed as a "project" using CodeWright; a four DSP application requires four projects.

For each project, we arrange all source files into a common directory. We then build a source file which contains only a list of "#include" statements identifying all source and header files required for this executable (see Appendix A). This "selector" file is added to the "member list" for this project.

A linker command file (iiq67.cmd) has been built for 3DMD. It specifies the memory map for each DSP. It is unlikely that this map will have to be changed.

There are a number of steps required to build a new project.
1. Create a directory structure (see above).
2. Copy an existing file structure into the new directory, if possible.
3. Edit the copied files, or build new files as necessary for the new project. The "selector" source file must be edited.
4. Use CodeWright to create a new project within the new directory path.
5. Choose the "selector" file and the linker command file as the only "project members".
6. Build the application.

7. How to Run the Demonstration

This section explains how to load and run the 3DMD demonstration. It assumes that the hardware is fully configured. The steps are listed below. Note that all interactions occur on the Q67 PC. The M62 is only used for software development.

Precaution

The PCs must be powered up *before* external signals are applied to the embedded DSP cards. This is essential in order to avoid damage to the interface hardware. Furthermore, external signals to the embedded DSPs must be turned off *before* power to the PC is shut down.

Turn on the Hardware

- Turn on both PCs and allow to boot.
- Switch on the power supply for the Camera Interface Card and the Clock Generator Card.
- Switch on the power for the synchronisation logic.
- Turn on the power for the 2 galvanometer controllers.
- Turn on the laser.

Initialise the DSPs

- On Q67 PC desktop, double click the *boot M6x* icon
- Double click the *qboot Q67* icon. Type "4" in the DOS box that opens to initialise all 4 DSPs.
- Double click *Q67 JTAGdiag*, to launch the JTAG diagnostic. Drag the horizontal scroll bar and click RESET then EXIT. This initialises the JTAG debugger.

Launch the M62 Terminal emulators

- Double click *UniTerminal M62_0 'Galvo'* icon.
- Double click *UniTerminal M62_1 'Peak'* icon.
- Click the "Boot target DSP" button on each instance of UniTerminal.

Load the M62 DSPs

The M62 processors are loaded directly using the UniTerminal utility. Files are located on the M62 PC and are accessible across the network. Since these are not MQX applications, they can be loaded directly into on-chip memory and executed without the use of the debugger and Code Composer Studio.

- On the pull-down menu, select File->Coff file->Download.
- Navigate to Network Neighbourhood->Entire Network->VIT->M62pc->C->Working->WaveformDemos->Demo5->Peak5->iim62->inc->z116.
- Double click Peak5.out
- Alternatively, once a path has been selected, simply clicking the "Re-execute Coff File" button will reload and start the application immediately.
- The file loads and runs. The message "Peak5" followed by some debug information should appear on the terminal emulator.
- Similarly, navigate to the Galvo5 directory and double click Galvo5.out
- The file loads and runs. The message "Galvo5" followed by some debug information should appear.

Load the Visual Basic User Interface Application

- Double click the *VisualBasic6* icon to start Visual Basic.
- Select the "motionb0 project"
- Minimise the window

Load CCS

Note: if at anytime CCS fails to boot or hangs during bootup, abort the application, reboot the Q67 DSP by double clicking *qboot Q67* as described above and reboot the JTAG debugger by clicking *Q67 JTAGdiag* and clicking "Reset" as described above.

- Double click the *CCStudio* icon.
- 4 windows should open: the "Parallel Debug Manager" and 3 debug windows, one for each DSP that is used.

Load the Q67 DSPs

Loading must be done via CCS.

- From a debug window, use the pull-down menu to select:
  GEL -> My Functions -> Load
- Click to download that processor.
- Repeat for the remaining 2 processors.

If at anytime the system hangs or reports an error, terminate CCS and repeat the instructions above (under "Load CCS") to reboot the DSP tools.

Start the Application

- In the "Parallel Debug Manager", click "Run".
- Maximise the Visual Basic window and click the run icon.

Run the Application

Once launched, the user interface application presents a window labelled "Waveform Controller". This is the main window for the user interface.

- Click the "Start Target" command button and wait for the "Waveform controller ready" message to appear in the "Target Response" display box.
- Click "Init Target". The command buttons in the "Wave Generator" frame should become active in addition to the remaining buttons in the "Main" frame.

- Click “Go” to start the demo and “Stop” to terminate it.

Once the system is running, the laser pattern should be projected against a white background set about 45 cm in front of the head. Place a small white target in front of the background and move it through the scanned pattern. The large display box in the user interface will show the position of the object as it moves about, Fig. 20. Beside this box, a narrow vertical display box shows the magnitude of the movement (higher is faster). Note that if the system were calibrated, quantitative values of range and position could be obtained.

Move the entire white background object back and forth and note that the entire scan pattern is visible in the display window.

If at anytime the system hangs, Visual Basic will have to be terminated by typing the Ctrl/Alt/Del key combination and closing the application. CCS should be closed down,

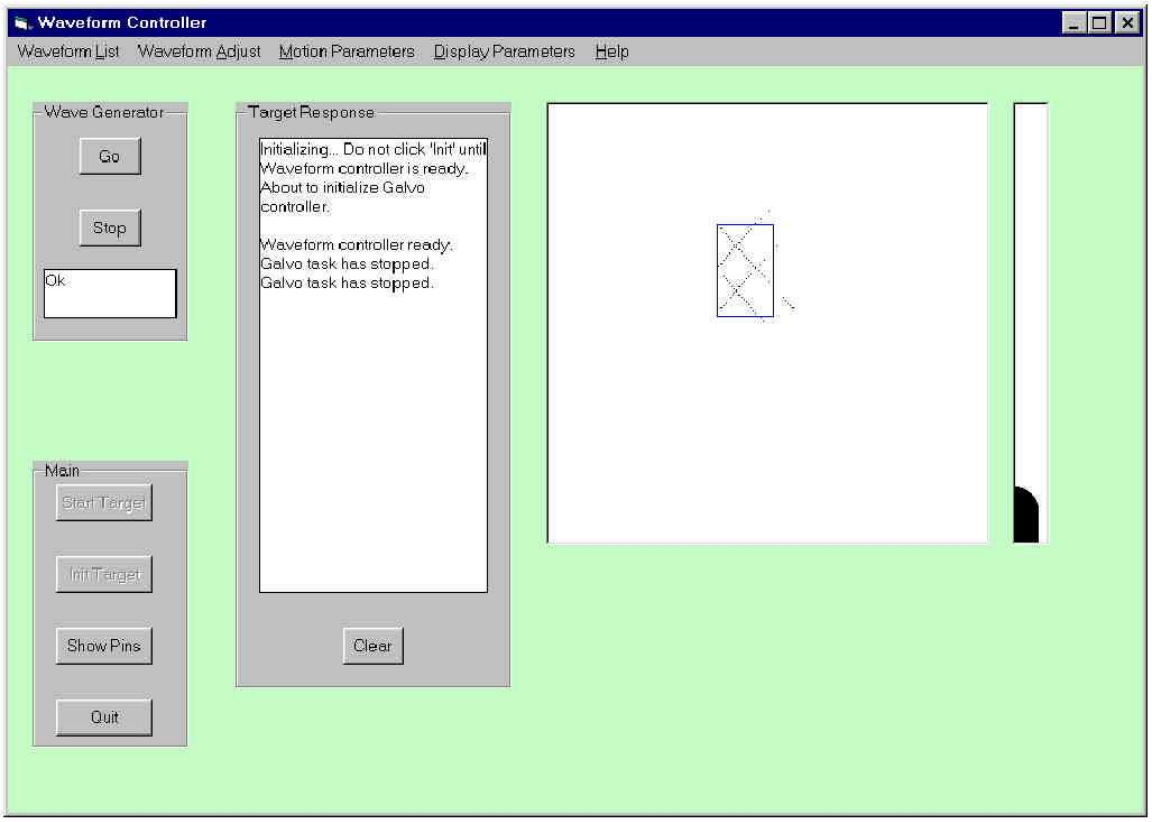

Figure 20. 3DMD Capturing the Motion of a Small Object.

The position of an object is displayed dynamically as it moves about within the field-of-view of the scanner.

and all M62 processors reset and reloaded. Q67 and JTAG interface should be rebooted and the entire process repeated. These development tools have always been very fragile.

Note that the M62 applications can be quickly reloaded by clicking the "lightening bolt" button on the toolbar.

8. Results

This section presents some early performance results for 3DMD. The principal factor that limits system performance is the sample rate of the prototype scanner. It generates range samples at a rate of 25.6 µs per point, or at about 39 kHz. The length of each scan line therefore determines the update rate of the motion detector. For example, the duration of a scan 'line', i.e. a single sweep of the scanning waveform, consisting of 2048 points is 52.4 ms which corresponds to a motion detection rate of 19 Hz. Similarly, a 512 point scan line will permit a motion detection rate of 76.3 Hz. These are examples of the actual performance of the motion detector. A secondary performance factor is the display rate of the user interface. The update rate of the motion detector is independent of the data display rate. We have not attempted to measure the performance of the display system, but we estimate an update rate of about 5 – 10 Hz.

Spatial volume sampling density is directly related to the frequency of the scanning waveform. For the prototype scanner, a number of factors limit the operating volume of the motion detector. These factors include the galvanometer bandwidth, mirror inertia, the optical field-of-view and the laser power. Driving the scanner at a lower scanning frequency (smaller number of cycles in the Lissajous pattern) produces a larger scanning cross-section. Table I demonstrates the relationship between scanning waveforms and the resulting maximum scanning volume. It is possible to circumvent this frequency response limitation by augmenting the drive amplitude of the Lissajous patterns (section 3.2). Sample results are plotted in Fig. 21.

X- Axis Cycles	Y- Axis Cycles	Total Cycles (X + Y)	Max Scan Width at mid-range ΔX (cm)	Max Scan Width at mid-range ΔY (cm)	Max Scan cross-sectional area (cm^2)	Max Active Volume (cm^3)
3	2	5	13.5	25.5	344.25	8606.25
4	3	7	9.8	25	245	6125
5	4	9	7.2	20.6	148.32	3708
6	5	11	5.3	20	106	2650

Table 1 3D Motion Detector Maximum Scanning Volume

The volume scanned by the prototype motion detector is affected by the bandwidth of the scanning galvanometers. For Lissajous scanning waveforms, as the number of cycles in a scan pattern increases (Col. 3), the bandwidth effects become apparent. This is especially noticeable in the faster X-axis (Col. 4). In its current configuration, the scanner can obtain range measurements from about 30 to 55 cm. The Active Volume (Col. 7) is determined by multiplying the scan cross-sectional area at mid-range (Col. 6), by the depth-of-field of the sensor, 25 cm. A 2048-point scan line operating at a 19 Hz repetition rate was used for these measurements.

Jitter

The exact relationship between the CCD sampling process and the galvo position control process is determined largely by the software behaviour of the DSPs and especially by the interrupt performance of the galvo control DSP. This relationship is fundamental; large variations can lead to positional uncertainty that compromises system accuracy.

For the prototype system, we have measured the jitter in waveform generation to be about ±220 ns. Since range samples are generated at a fixed rate of 25.6 µs, this positional uncertainty corresponds to about ± 0.22/25.6 = ± 1/116 of a voxel. This level of jitter is well below the system noise level and can be safely ignored.

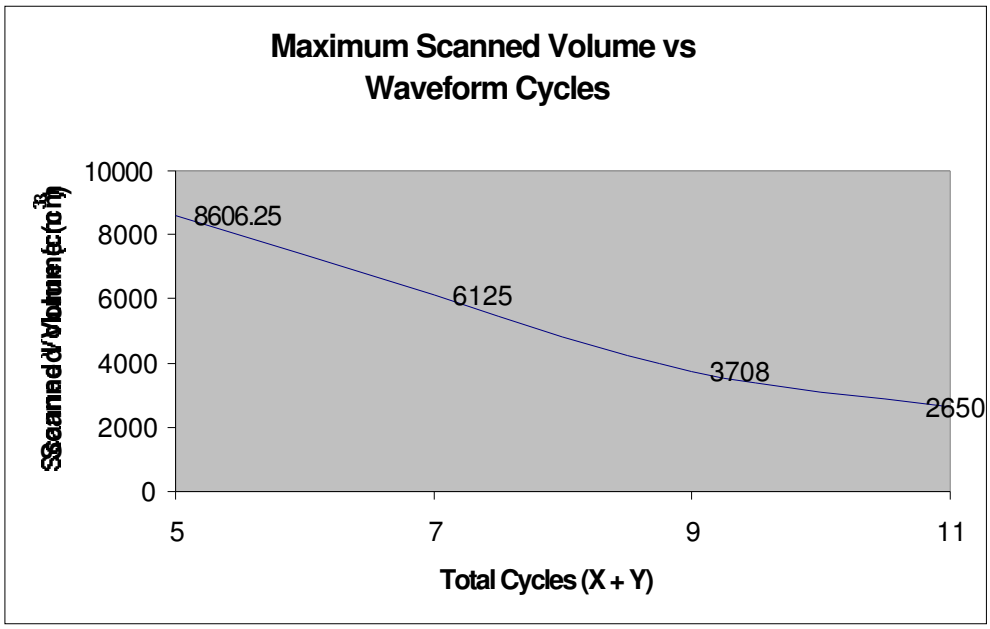

Figure 21. Maximum Scanning Volume of the 3D Motion Detector.

Total Cycles is the total number of cycles in the Lissajous Scanning Waveform and illustrates the effect of limitations in galvanometer bandwidth. The data presented here is listed in Table I.

9. Conclusions

This report has described the development of an experimental motion detector based on a high precision 3D auto-synchronised laser range scanner. With the current optical head we used, the prototype system can scan a maximum volume of about 9000 cm^3. Hovever

other prototypes of this same optical principle can be used to expand the measurement volume to several m^3.

A compromise exists between maximum scanning volume, scanning density and scanning speed. This is due largely to the inertia of the galvanometer/mirror scanning devices. Large scanning volumes usually require slower galvanometer scan rates which can be obtained using either small cycle Lissajous cycles (e.g. 3:2), or a higher point density per scan. Maximum scanning speed is determined by the line rate of the scanner that is dependent on the number of points per scan line.

For example, a 19 Hz repetition rate can be achieved using a 2048-point scan line or a 9.5 Hz repitition rate can be obtained with a 4096-point scan line. In the later case, a higher density Lissajous pattern (e.g. 8:7) can be used. When calibrated, 3DMD could perform quantitative motion measurements.

3DMD is based on the MDSP system architecture. This architecture supports the development of extendible real-time systems through a combination of hardware and software modularity.

Commercial DSP hardware components and a commercial real-time operating system were used in the development of 3DMD. This project relied heavily on newly released hardware based on early releases of commercial DSP chips. System development was affected by problems that can be traced to the unproven state of these products. This included faulty hardware, limitations in processor and/or board functionality and major errors in runtime support libraries. Identifying and correcting many of these problems was expensive.

10. References

1. F. Blais, M. Rioux, and J.-A. Beraldin, “Practical considerations for a design of a high precision 3-D laser scanner system,” *SPIE Proceedings*, Vol. 959, pp. 225-246, June 28-29, 1988.

2. J. Taylor, J.-A. Beraldin, and G. Godin, "3D Imaging collaboration between the National Research Council of Canada and European Museums and Cultural Organizations", *Proceedings of the Electronic Imaging and the Visual Arts (EVA 2001)*, Montreal,Que.. October 3-5, 2001 NRC 44897.

3. M. Rioux, “Automatic systems for the identification and inspection of humans”, *SPIE Proceedings*, Vol. 2277. pp. 42-54, July 28-29, 1994 NRC 3834.

4. M. Rioux, “Laser range finder based on synchronized scanners”, *SPIE Milestone Series, Optical Techniques for Industrial Inspection*, Paolo Cielo and Brian J. Thompson (Editors). Vol. MS 135. SPIE Optical Engineering Press, Bellingham WA. 1997. pp. 142-149 NRC 40222.

5. J.-A. Beraldin, F. Blais, M. Rioux, L. Cournoyer, D. Laurin, and S.G. MacLean, “Eye-safe digital 3D sensing for space applications,” *Opt. Eng*. **39**(1), pp. 196-211, 2000.

6. F. Blais, M. Rioux, and S.G. Maclean, "Intelligent, variable resolution laser scanner for the space vision system", "*SPIE Proceedings Acquisition, Tracking, and Pointing V*", Vol. 1482, pp.473-479, April 3-5, 1991, NRC 31808.

7. D. Green, F. Blais, J.-A. Beraldin, and L. Cournoyer, "MDSP: a modular DSP architecture for a real-time 3D laser range sensor*", "Proceedings SPIE Three Dimensional Image Capture and Applications V*", Vol. 4661, pp. 9-19, Jan. 23-24 2002, NRC 44896.

8. http://www.psti.com/

9. F. Blais, and M. Rioux, "Real-time numerical peak detector", *Signal Processing 11*, pp. 145-155, 1986.

10. http://www.innovative-dsp.com/

11. "DSPComponent User's Manual", Rev. 1.01, Innovative Integration Inc. March 22 1999.

12. "Precise/MQX Version 2.40 User's Guide", Precise Software Technologies Inc., July 1999.

13. http://dspvillage.ti.com/

14. D.A. Green, F. Blais, M. Sauks, "Interprocessor message passing performance issues for multiple DSP-based applications using a real-time operating system", IEEE Micro (submitted June 2001).

15. Private communication, Innovative Integration Inc.

16. "Quatro62 Development Package Software Manual", rev. 1.00, Innovative Integration Inc. 1998.

11. Appendix

The Appendix is divided into four sections:

A. Listings of DSP software that runs under MQX,
B. Listings of Peak DSP and Galvo DSP stand-alone software,
C. Listings of user interface software written in Visual Basic, and
D. Design note discussing the installation and initial port of MQX to Q67 and M62 DSP platforms.

Appendix A. Code Listings of Real-time DSP Software.

The 3DMD application is distributed across 3 DSPs on a Q67 board and 2 additional DSPs on M62 cards. Appendix A lists the Q67 software. It is composed of 3 executable images called demo0A, demo0B and demo0C. The listings for each processor begin with its "selector" file (code67A.c, code67B.c and code67C.c) that identifies all source files to be compiled into that executable image. Following the selector files for processors B and C, we list only the source files that have not already appeared. Only application "include" files that we have developed are presented here, MQX and libraries provided by the board manufacturer are proprietary. At the end, we list the linker command file that specifies the memory map for the DSPs and the "*.mki make include files" which are used by CodeWright to automatically generate the "makefiles".

code67A.c

```
/* IIQ67 includes                          */
# include "c:\q6x\include\target\periph.h"
# include "c:\q6x\include\target\stdio.h"

/* MQX includes */
# include "c:\precise\mqx2.40\lib\3206701b.ti\mqx.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\fio.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\message.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\32060.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\mqx_prv.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\ipc.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\ipc_prv.h"
# include "c:\precise\mqx2.40\lib\iiq67b.ti\ipc_prv2.h"
# include "c:\precise\mqx2.40\lib\iiq67b.ti\ipcfifo1.h"
# include "c:\precise\mqx2.40\lib\iiq67b.ti\bsp.h"


/* application includes */
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\demodefs.h"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\demotyps.h"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\hardware.h"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\mboxput.c"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\mboxget.c"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\mboxprintf.c"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\mboxscanf.c"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\demo0A.c"
# include “c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\displaydriver.c"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\mboxdisplaydat.c"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\user.c"

/*-----------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*----------------------------------------------------------------------*/
```

demodefs.h

```
#ifndef  _demo_defs_h_
#define  _demo_defs_h_

#define full_version 0    // #ifdef used for early development

/* Global indicies: task template indecies. */
#define IPC_TTN                    10
#define USER_INTERFACE_TASK        11
#define GALVO_CONTROL_TASK         15
#define PEAK_DATA_TASK             20
#define FILE_HANDLER_TASK          25
#define GALVO_DATA_TASK            30
#define MOTION_DETECT_TASK         35
#define DISPLAY_DRIVER_TASK        40

/* Queues - <#1 - 7 are reserved> */
#define USER_INTERFACE_Q           40
#define GALVO_CONTROL_Q            41
#define GALVO_DATA_Q               42
#define PEAK_DATA_Q                43
#define FILE_HANDLER_Q             44
#define PEAK_NOTIFIER_SYS_Q        45
#define MOTION_DETECT_Q            46
#define DISPLAY_DRIVER_Q           47
#define MOTION_DETECT_2_Q          48
#define GALVO_NOTIFIER_SYS_Q       49

/* Interprocessor message queues */
/* Q67 FIFOLinks - Board 1 */
// ProcA
#define    FLINK1_A_B_Q       10
#define    FLINK1_A_C_Q       11
#define    FLINK1_A_D_Q       12
// ProcB
#define    FLINK1_B_A_Q       13
#define    FLINK1_B_C_Q       14
#define    FLINK1_B_D_Q       15
// ProcC
#define    FLINK1_C_A_Q       16
#define    FLINK1_C_B_Q       17
#define    FLINK1_C_D_Q       18
// ProcD
#define    FLINK1_D_A_Q       19
#define    FLINK1_D_B_Q       20
#define    FLINK1_D_C_Q       21

/* Q67 FIFOLinks - Board 2 (future expansion) */
// ProcA
#define    FLINK2_A_B_Q       22
#define    FLINK2_A_C_Q       23
#define    FLINK2_A_D_Q       24
// ProcB
#define    FLINK2_B_A_Q       25
#define    FLINK2_B_C_Q       26
#define    FLINK2_B_D_Q       27
// ProcC
#define    FLINK2_C_A_Q       28
#define    FLINK2_C_B_Q       29
#define    FLINK2_C_D_Q       30
// ProcD
#define    FLINK2_D_A_Q       31
#define    FLINK2_D_B_Q       32
#define    FLINK2_D_C_Q       33
```

demodefs.h continued

```
/* Processor numbers */
/* Q67 Board 1 */
#define Q67_1A_ID          1
#define Q67_1B_ID          2
#define Q67_1C_ID          3
#define Q67_1D_ID          4

/* Q67 Board 2 (future expansion) */
#define Q67_2A_ID          5
#define Q67_2B_ID          6
#define Q67_2C_ID          7
#define Q67_2D_ID          8

/* M62 Boards (NOT REQUIRED) */
#define M62_1_ID           101    // galvo processor
#define M62_2_ID           102    // future expansion

/* misc */
#define On_This_Processor  0
#define MAX_BUFFERS        16

/* DMA related */
#define DMA_CH_3           3 // DMA channel for FIFOPort access
#define BLOCK              1
#define NO_BLOCK           0

/* Galvo command types */
#define INIT_GALVO                0x1122
#define STOP_GALVO                0x1010
#define START_GALVO               0x2020
#define DEFINE_WAVE               0x3030
#define NEW_AMP                   0x4040
#define NEW_PHASE                 0x5050
#define NEW_OFFSET                0x6060
#define CHANGE_WAVEFORM           0x7070
#define TEST_CASE                 0x1080
#define PRINT_VALUES              0x1090

/* Galvo misc */
#define GALVO_ACK                 0x4040
#define GALVO_OK                  0x1111
#define BAD_GALVO_RESPONSE        0x2222
#define NO_GALVO_RESPONSE         0x3333
#define GALVO_NOT_READY           0x4444

// Waveform types
#define COS                1
#define SAWTOOTH           1
#define UNDEFINED          2

// Waveform misc
#define MAX_NUM_WAVEFORMS        16
#define MAX_AXES                 2
#define MAX_VOXELS_PER_LINE      0x1000    // 4096 voxels/line max.
#define X_AXIS                   0
#define Y_AXIS                   1

/* Galvo run mode */
#define CONTINUOUS_MODE          0
#define IMAGE_MODE               0x1
```

demodefs.h continued

```
/* Peak commands */
#define START_PEAK          0x5050
#define RQST_LINE           0x6060

/* Peak misc. */
#define INIT_FPORT          0x5555
#define PEAK_READY          0x1010

/* No. of Peaks to Detect, for colour systems */
#define NUMBER_OF_PEAKS       0x1

/* Peak Buffer information */
#define PEAK_BUFFER_SIZE      0x00800000  // 8 Mbytes = 2Mpeaks (@4 bytes/peak)
#define MAX_NUM_OF_VOXELS     PEAK_BUFFER_SIZE / (4 * NUMBER_OF_PEAKS) // (@ 4 bytes/peak)

/* Motion detector information */
#define MAX_CHANGED_VOXELS_PER_LINE  0x1000    // limits message size
#define MOTION_K_FACTOR              0.2       // filter constant
#define MOTION_DIFF_THRESHOLD        3         // (CCD pixels)
#define MOTION_DIR_POS               0x0
#define MOTION_DIR_NEG               0x1
#define MOTION_DIR_BOTH              0x2

/* Display Driver message types */
#define MOTION          0x11
#define GALVO           0x22

/* FIFOPort information */
#define FIFOPORT_BUFFER_SIZE      0x200

/* Galvo buffer information */
#define MAX_BUFFERED_LINES     16 // num lines buffered (for galvo position)
#define SAMPLES_PER_PACKET     FIFOPORT_BUFFER_SIZE/(2 * 2)  // half buffer, 2 int_16s per sample

/* File handler message types */
#define OPEN_FILE           0x0
#define WRITE_DATA          0x1
#define CLOSE_ALL_FILES     0x2

/* File handler data types */
#define POSITION            0x0
#define INTENSITY           0x1

/* file status */
#define FILES_OPEN          0x1
#define FILES_CLOSED        0x0

/* Utility message Data types */
#define NO_WRITE            0x1
#define OPEN_FAIL           0x2

/* Display driver task message types */
#define RQST_NEXT_DATA      0x1

/* Mailbox communications: Commands, Host to Target */
#define WVFRM_SHOW_PINS        1
#define WVFRM_SHOW_PARAMS      2
#define WVFRM_DEFINE           3
#define WVFRM_EDIT             4
#define WVFRM_SELECT           5
#define WVFRM_TEST_COMM        6
#define WVFRM_GO               7
#define WVFRM_STOP             8
```

demodefs.h continued

```
#define WVFRM_QUIT                  9
#define LAUNCH_TARGET               0xa
#define WVFRM_MOTION_PARAMS         0xb


/* waveform editing commands */
#define WVFRM_EDIT_AMPLITUDE        1
#define WVFRM_EDIT_OFFSET           2
#define WVFRM_EDIT_PHASE            3


/* Mailbox communications: Command responses, Target to Host */
#define CMD_OK                      0
#define CMD_REJECTED                1
#define CMD_UNKNOWN                 2

/* Mailbox communications: Data types for multiplexing. */
#define STRING                      0
#define GRAPHIC_POINT               1
#define GRAPHIC_LINE                2
#define GRAPHIC_CIRCLE              3
/* tbd */

#endif
/*-----------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------------*/
```

demotyps.h

```
#ifndef  _demo_typs_h_
#define  _demo_typs_h_

typedef volatile struct scan_info_struct
{
    uint_32    WAVE_TYPE;          // cos, sawtooth
    float      AMP;                // amplitude in volts
    float      OFFSET;             // offset in volts
    float      PHASE;              // radians
    float      PHASE_CORRECTION;   // radians
    uint_16    VOXELS_PER_LINE;
    uint_16    CYCLES;             // cycles per line (lissajous)
    uint_16    RASTER_INC;         // incr for this axis, triggered by other axis
    uint_16    RESERVED;           // for quadbyte alignment
} SCAN_INFO_STRUCT, * SCAN_INFO_STRUCT_PTR;


typedef volatile struct waveform_define_struct
{
    SCAN_INFO_STRUCTSCAN_INFO_X;      // x-axis waveform
    SCAN_INFO_STRUCTSCAN_INFO_Y;      // y-axis waveform
} WAVEFORM_DEFINE_STRUCT, * WAVEFORM_DEFINE_STRUCT_PTR;


typedef volatile struct galvo_cmd_message
{
    MESSAGE_HEADER_STRUCT     HEADER;
    uint_16                   GALVO_CMD;
    uint_16                   WAVE_INDEX;
    WAVEFORM_DEFINE_STRUCT    WAVEFORM_DEF;
} GALVO_CMD_MESSAGE_STRUCT, * GALVO_CMD_MESSAGE_STRUCT_PTR;


// used for low-level fifoport command transfers to galvo processor
typedef volatile struct galvo_cmd_transfer_struct
{
    uint_16          GALVO_CMD;     // fixed single transfers
} GALVO_CMD_TRANSFER_STRUCT, * GALVO_CMD_TRANSFER_STRUCT_PTR;


// used for low-level waveform definition transfers to galvo processor
typedef volatile struct galvo_waveform_transfer_struct
{
    uint_16          GALVO_CMD;     // long transfers
    uint_16          WAVE_INDEX;
    uint_16          TRANSFER_SIZE;
    uint_16          RESERVED;      // for alignment
    WAVEFORM_DEFINE_STRUCT    WAVEFORM_DEF;
} GALVO_WAVEFORM_TRANSFER_STRUCT, * GALVO_WAVEFORM_TRANSFER_STRUCT_PTR;


typedef volatile struct waveform_master_struct
{
    uint_16                   WAVE_INDEX;
    uint_16                   RESERVED;
    WAVEFORM_DEFINE_STRUCT    WAVEFORM[MAX_NUM_WAVEFORMS];
} WAVEFORM_MASTER_STRUCT, * WAVEFORM_MASTER_STRUCT_PTR;


typedef volatile struct peak_cmd_message
{
    MESSAGE_HEADER_STRUCT     HEADER;
    uint_32                   CMD;
    uint_32                   VOXELS_PER_LINE;
```

demotyps.h continued

```
    uint_32                         LINE_NUM;
    uint_32                         MOTION_DIRECTION;
    float                           MOTION_FILTER_CONSTANT;
    uint_32                         MOTION_DETECT_THRESHOLD;
} PEAK_CMD_MESSAGE, * PEAK_CMD_MESSAGE_PTR;


typedef volatile struct utility_message
{
    MESSAGE_HEADER_STRUCT     HEADER;
    uint_32                         DATA;
} UTILITY_MESSAGE, *UTILITY_MESSAGE_PTR;


typedef struct peak_data
{
    uint_16             PEAK_POSN;
    uint_16             PEAK_INT;
} PEAK_DATA, * PEAK_DATA_PTR;


typedef struct voxel_dat
{
    PEAK_DATA         VOXEL[NUMBER_OF_PEAKS];
} VOXEL_DATA, * VOXEL_DATA_PTR;


typedef union
{
    uint_16             ALL_BUFF[PEAK_BUFFER_SIZE/2];
    VOXEL_DATA        VOXEL_BUFF[MAX_NUM_OF_VOXELS];
} PEAK_DATA_BUFFER, * PEAK_DATA_BUFFER_PTR;


typedef volatile struct info_struct
{
    uint_32             MODE;                           // image/continuous
    uint_32             VOXEL_CNTR;
    uint_32             VOXELS_PER_LINE;
    uint_32             TOTAL_VOXELS;
    uint_32             LINE_NUM;
    uint_32             COLD_START;
    uint_32             WARM_START;
    uint_32             FIRST_PASS;
    uint_32             MOTION_DIRECTION;               // pos/neg/both
    float               MOTION_FILTER_CONSTANT;   // 'k'
    uint_32             MOTION_DETECT_THRESHOLD;
    uint_16             *BUFFER_PTR;
    uint_16             *REF_BUFFER_PTR;
} PEAK_DATA_INFO_STRUCT, * PEAK_DATA_INFO_STRUCT_PTR;


// struct contains MAX_BUFFERED_LINES, each of max length MAX_VOXELS_PER_LINE
// data are interleaved as int_16s.
typedef union galvo_position
{
    int_32 volatile     BOTH;

    struct
    {
        int_32 volatile     X_GALVO_POSN  : 16;
        int_32 volatile     Y_GALVO_POSN  : 16;
    } PAIR;
```

demotyps.h continued

```
} GALVO_POSN_DATA, * GALVO_POSN_DATA_PTR;

typedef struct galvo_intermediate_struct
{
    GALVO_POSN_DATA          GALVO_DATA_ARRAY[MAX_BUFFERED_LINES][MAX_VOXELS_PER_LINE];
} GALVO_INT_STRUCT, * GALVO_INT_STRUCT_PTR;

typedef struct galvo_info_struct
{
    uint_32                  LINE_INDEX;
    uint_32                  POINT_INDEX;
    uint_32                  VOXELS_PER_LINE;
    GALVO_INT_STRUCT_PTR     GALVO_DATA_PTR;
} GALVO_DATA_INFO_STRUCT, * GALVO_DATA_INFO_STRUCT_PTR;

typedef struct file_handler_message
{
    MESSAGE_HEADER_STRUCT    HEADER;
    uint_32                  FILENAME_POS[128];
    uint_32                  FILENAME_INT[128];
    uint_32                  FILENAME_X[128];
    uint_32                  TYPE;
    uint_32                  DATA_TYPE;
    uint_32                  ELEMENTS_PER_LINE;
    uint_32                  NUM_LINES;
    uint_32                  DECIMATION;
} FILE_HANDLER_MESSAGE, *FILE_HANDLER_MESSAGE_PTR;


typedef struct peak_data_rqst_msg
{
    MESSAGE_HEADER_STRUCT    HEADER;
    uint_32                  TYPE;
    uint_32                  LINE_NUM;
} PEAK_DATA_RQST_MSG, *PEAK_DATA_RQST_MSG_PTR;


typedef struct peak_data_rply_msg
{
    MESSAGE_HEADER_STRUCT    HEADER;
    VOXEL_DATA               VOXEL_BUFF[MAX_VOXELS_PER_LINE];
} PEAK_DATA_RPLY_MSG, *PEAK_DATA_RPLY_MSG_PTR;

typedef struct line_info_msg
{
    MESSAGE_HEADER_STRUCT          HEADER;
    PEAK_DATA_INFO_STRUCT_PTR      INFO_STRUCT_PTR;
} LINE_INFO_MSG, *LINE_INFO_MSG_PTR;

typedef struct display_driver_data_struct
{
    uint_32      COUNT;
    uint_16      INDEX[MAX_CHANGED_VOXELS_PER_LINE];
    int_16       VALUE[MAX_CHANGED_VOXELS_PER_LINE];
    int_16       X_GALVO_POS[MAX_CHANGED_VOXELS_PER_LINE];
    int_16       Y_GALVO_POS[MAX_CHANGED_VOXELS_PER_LINE];
} DISPLAY_DRIVER_DATA_STRUCT, *DISPLAY_DRIVER_DATA_STRUCT_PTR;

typedef struct display_driver_msg
{
    MESSAGE_HEADER_STRUCT        HEADER;
    uint_32                      TYPE;      //motion or galvo
    DISPLAY_DRIVER_DATA_STRUCT   DATA;
```

demotyps.h continued

```
} DISPLAY_DRIVER_MSG, *DISPLAY_DRIVER_MSG_PTR;


/* Prototypes */
extern void User_Interface_Task(uint_32 index);
extern void Peak_Data_Task(uint_32 index);
extern void Galvo_Control_Task(uint_32 index);
extern void Galvo_Data_Notifier(GALVO_DATA_INFO_STRUCT_PTR info_ptr);
extern void Motion_Detect_Task(uint_32 parameter);
extern void Display_Driver_Task(uint_32 parameter);

extern uint_32    check_galvo_response(void);

// MailBox functions
extern uint_32 mbox_get(void);
extern void mbox_put(uint_32);
extern uint_32 mbox_printf(char *, ...);
extern uint_32 mbox_display_data( DISPLAY_DRIVER_DATA_STRUCT_PTR );


#endif
/*-----------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------------*/
```

hardware.h

```
#ifndef  _hardware_h_
#define  _hardware_h_

#define   MAX_PEAK_DATA_BUFF_SIZE  0x10000
#define   MAX_PEAK_DATA_BUFF_ENTRIES  MAX_PEAK_DATA_BUFF_SIZE / 4  /* tbd */

#endif

/*-----------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------*/
```

mboxput.c

```
/*****************************************************************
 *
 * mbox_put()
 *
 *  FILENAME: mboxput.c
 *
 *  PARAMETERS:
 *          none
 *
 *  DESCRIPTION:
 *          writes a single uint_32 to the PCI mailbox.
 *
 *          D.Green
 *          2001-07-06
 *
 *  RETURNS:
 *          none
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *
 *****************************************************************/



void mbox_put(uint_32 value)
{
    //check status using blocking delay to allow other tasks to run
    while (check_outbox(0) == 0)  // wait while full
        _time_delay(1);       //block for 1 ms

    write_mailbox(value, 0);

} //Endbody
```

mboxget.c

```
/*****************************************************************
 *
 * mbox_get()
 *
 *  FILENAME: mboxget.c
 *
 *  PARAMETERS:
 *          none
 *
 *  DESCRIPTION:
 *          fetches a single uint_32 from the PCI mailbox.
 *
 *          D.Green
 *          2001-07-06
 *
 *  RETURNS:
 *          uint_32
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *
 *****************************************************************/



uint_32 mbox_get(void)
{
    //check status using blocking delay to allow other tasks to run
    while (check_inbox(0) == 0)    // wait while empty
        _time_delay(1);       //block for 1 ms

    return(read_mailbox(0));

} //Endbody
```

mboxprintf.c

```
/*****************************************************************
 *
 * mbox_printf()
 *
 *  FILENAME: mboxprintf.c
 *
 *  PARAMETERS:
 *                pointer to printf-style format string for transfer to mailbox
 *                variable argument list as used in printf() function.
 *
 *  DESCRIPTION:
 *                writes a formatted string to mailbox for transfer to host PC
 *
 *                When        Who       What
 *                2001-07-11  D.G       renamed from mboxputstr.c
 *                2001-08-27  D.G.      added TYPE field to support multiplexing
 *                                      the mailbox
 *
 *
 *  RETURNS:
 *          number of characters transferred, or 0 if failed.
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *
 *****************************************************************/


uint_32 mbox_printf(char *format, ...)
{
    uint_32       length;
    uint_32       i, j;
    uint_32       temp_string[128]; // max string length
    uint_32       out_string[128];
    va_list       arg_ptr;

    va_start(arg_ptr, format);
    // build a packed formatted string in 'temp_string'
    length = vsprintf((char_ptr)&temp_string, format, arg_ptr);
    va_end(arg_ptr);

    if (length == 0)
        return(0); // return if 0 length

    // unpack the string (easier here than in Visual Basic)
    for (i = 0; i <= length/4; i++ )
    {
        j = i * 4;
        out_string[j] = (temp_string[i] & 0xff000000) >> 24;
        out_string[j+1] = (temp_string[i] & 0xff0000) >> 16;
        out_string[j+2] = (temp_string[i] & 0xff00) >> 8;
        out_string[j+3] = (temp_string[i] & 0xff);
    }

    //check status using blocking delay to allow other tasks run
    while (check_outbox(0) == 0) // wait while full
        _time_delay(1); //block for 1 ms

    //remaining transfers don't delay
    //send length of transfer via interrupt
    mailbox_interrupt( length );

    // send message TYPE (String, graphic)
    write_mailbox( STRING, 0 );
```

mboxprintf.c continued

```
    //one transfer per char!! not efficient but ok for short messages.
    for (i = 0; i < length; ++i)
        write_mailbox( out_string[i], 0 );

    return(length);

} //Endbody
```

mboxscanf.c

```
/*****************************************************************
 *
 *  mbox_scanf()
 *
 *  FILENAME: mboxscanf.c
 *
 *  PARAMETERS:
 *                pointer to printf-style format string for transfer from mailbox
 *                variable argument list as used in scanf() function.
 *
 *  DESCRIPTION:
 *                reads a formatted string from mailbox (transfer from host)
 *
 *                When        Who       What
 *                2001-07-13  D.G       first version
 *
 *  RETURNS:
 *                number of conversions made    (success)
 *                IO_EOF i.e. -1            (failure)
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *
 *****************************************************************/

int_32 mbox_scanf(char *format, ...)

{   //Body

    va_list       arg_ptr;
    char          temp_string[132];
    uint_32       length, i;
    uint_32       result;

    // fetch string length
    length = mbox_get();

    // incoming data is an ASCII string
    // fetch string char by char and build string buffer
    for (i = 0; i < length; i++ )
        temp_string[i] = (char)mbox_get();

    temp_string[length] = '\0';  // add null termination

    // do format conversion and variable assignment
    va_start(arg_ptr, format);
    result = _io_scanline(temp_string, format, arg_ptr);
    va_end(arg_ptr);
    return result;

 } //Endbody

//EOF
```

demo0A.c

```
//extern void User_Interface_task(uint_32 parameter);

TASK_TEMPLATE_STRUCT  MQX_template_list[] =
{
  { IPC_TTN,  _ipc_task, IPC_DEFAULT_STACK_SIZE, 0L, "_ipc_task", MQX_AUTO_START_TASK,   0L,
        0L},
  { USER_INTERFACE_TASK,    User_Interface_Task,8000L,1L, "User",
        MQX_AUTO_START_TASK | MQX_FLOATING_POINT_TASK,  0L, 0L},
  { DISPLAY_DRIVER_TASK,   Display_Driver_Task,  3500L, 2L, "display_driver", 0, 0L, 0L},
  { 0L,0L, 0L,0L, 0L, 0L,0L, 0L}
};


MQX_INITIALIZATION_STRUCT  MQX_init_struct =
{
  /* PROCESSOR_NUMBER */                  Q67_1A_ID,
  /* START_OF_KERNEL_MEMORY */            (pointer)(0x02700000),
  /* END_OF_KERNEL_MEMORY */              (pointer)(0x02fffff),
  /* INTERRUPT_STACK_SIZE */              BSP_DEFAULT_INTERRUPT_STACK_SIZE,
  /* TASK_TEMPLATE_LIST */                (pointer)MQX_template_list,
  /* MQX_HARDWARE_INT_LEVEL_MAX*/         BSP_DEFAULT_MQX_HARDWARE_INTERRUPT_LEVEL_MAX,
  /* MAX_MSGPOOLS */                      BSP_DEFAULT_MAX_MSGPOOLS,
  /* MAX_MSGQS */                         BSP_DEFAULT_MAX_MSGQS,
  /* IO_CHANNEL */                        BSP_DEFAULT_IO_CHANNEL,
  /* IO_OPEN_MODE */                      BSP_DEFAULT_IO_OPEN_MODE
};


/*
    To avoid confusion, processors are referred to as
    Q67_A_ID etc. Since MQX refers to proc1-4
    and Innovative Integration refers to proc0-3.
*/

/* Off processor message routing <min, max, queue> */
IPC_ROUTING_STRUCT _ipc_routing_table[] =
{
 {Q67_1B_ID, Q67_1B_ID, FLINK1_A_B_Q},     /* links from cpu_a to cpu_b */
 {Q67_1C_ID, Q67_1C_ID, FLINK1_A_C_Q},     /* links from cpu_a to cpu_c */
 {Q67_1D_ID, Q67_1D_ID, FLINK1_A_D_Q},     /* links from cpu_a to cpu_d */
 {0,0,0}
};

 IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_A_B =
 {
    /* IPC on ProcA uses FIFOLinkA_B to Q67 processor B */
    /* DMA_CHANNEL */                     0,
    /* MQX_DESTINATION_CPU */             Q67_1B_ID,
    /* IN_PACKET_MAX_SIZE   */            0x8500,
    /* IN_BUFFERS_TO_ALLOCATE */         4,
    /* IN_BUFFERS_TO_GROW */             2,
    /* IN_BUFFERS_MAX_ALLOCATE */        8,
    /* DEADLOCK_MASTER */                IPC_C6X_DEADLOCK_MASTER

 };

 IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_A_C =
 {
    /* IPC on ProcA uses FIFOLinkA_C to Q67 processor C */
    /* DMA_CHANNEL */                     1,
    /* MQX_DESTINATION_CPU */             Q67_1C_ID,
    /* IN_PACKET_MAX_SIZE */              0x8500, // large incoming data buffer transfers
    /* IN_BUFFERS_TO_ALLOCATE */         4,
    /* IN_BUFFERS_TO_GROW */             2,
```

demo0A.c continued

```
    /* IN_BUFFERS_MAX_ALLOCATE */     8,
    /* DEADLOCK_MASTER */             IPC_C6X_DEADLOCK_MASTER

 };

// CPU_D is not used in this demo
// IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_A_D =
// {
//   /* IPC on ProcA uses FIFOLinkA_D to Q67 processor D */
//   /* DMA_CHANNEL */                 2,
//   /* MQX_DESTINATION_CPU */         Q67_1D_ID,
//   /* IN_PACKET_MAX_SIZE */          512,
//   /* IN_BUFFERS_TO_ALLOCATE */      16,
//   /* IN_BUFFERS_TO_GROW */          4,
//   /* IN_BUFFERS_MAX_ALLOCATE */     28,
//   /* DEADLOCK_MASTER */             IPC_C6X_DEADLOCK_MASTER
//
// };

IPC_PROTOCOL_INIT_STRUCT _ipc_init_table[] =
{
  { (uint_32 (_CODE_PTR_) (IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))_IPC_FIFOLINK_channel_init,
    (pointer)&IPC_FIFO_1D_init_rec_A_B, "FIFOLink_channel_B", FLINK1_A_B_Q},
  { (uint_32 (_CODE_PTR_) (IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))_IPC_FIFOLINK_channel_init,
    (pointer)&IPC_FIFO_1D_init_rec_A_C, "FIFOLink_channel_C", FLINK1_A_C_Q},
//{ (uint_32 (_CODE_PTR_) (IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))_IPC_FIFOLINK_channel_init,
//   (pointer)&IPC_FIFO_1D_init_rec_A_D, "FIFOLink_channel_D", FLINK1_A_D_Q},
  {      NULL, NULL, NULL, 0}
};

/*-----------------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 1998-2001
*-----------------------------------------------------------------------------*/

/* EOF */
```

displaydriver.c

```
/*****************************************************************
 *
 *   DESCRIPTION:  Display Driver Task
 *
 *         This task is created by the Motion_Detect_Task.
 *         It receives messages from the Motion_Detect_Task
 *         containing points to be displayed in a VB
 *         picture box.
 *
 *         Messaging has been changed in this version (2001-11-07)
 *         In the original version, the display driver determined the
 *         rate at which range averaging took place.
 *         The new version changed this so that range data averaging
 *         now occurs in real-time. The display driver task
 *         operates at its own pace.
 *
 *         The addition of galvo position data requires that the
 *         display driver combine the 2 incoming data streams and
 *         integrate them into a single data struct.
 *
 *
 *
 *   AUTHOR:     D.Green
 *
 *   HISTORY:    First Version 2001-08-29
 *         When          Who   What
 *         2001-11-07    dg    changed messaging design, See above.
 *         2001-11-30    dg    added galvo position
 *
 *
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *****************************************************************/


/*TASK*-----------------------------------------------------------------
*
* Task Name : Display_Driver_Task
* Comments  :
*
*END*-----------------------------------------------------------------*/


void Display_Driver_Task
   (
      uint_32 parameter
   )

{ /* Body */

   _queue_id            display_driver_qid;
   _queue_id            motion_detect_qid;
   _queue_id            galvo_notifier_qid;

   uint_32              i;

   DISPLAY_DRIVER_MSG_PTR            display_driver_msg_ptr;
   DISPLAY_DRIVER_DATA_STRUCT_PTR display_driver_struct_ptr;
   _pool_id                          rqst_msg_pool_id;
   UTILITY_MESSAGE_PTR               rqst_msg_ptr, galvo_pos_rqst_msg_ptr;
   uint_32                           count;
```

displaydriver.c continued

```
    // open display driver message queue
    display_driver_qid = _msgq_open( DISPLAY_DRIVER_Q, 16L );
    if (display_driver_qid == 0)
            mbox_printf("Display_Driver_Task(): Failure in call _msg_open(). (code 0x%x)\n",
                _task_get_error() );

    // create message pool. use same pool for motion AND galvo request messages
    rqst_msg_pool_id = _msgpool_create( sizeof( UTILITY_MESSAGE ), 16L, 4L, 0L );
    if (rqst_msg_pool_id == 0)
        mbox_printf("Display_Driver_Task(): Failure in call to _msgpool_create().
            (code 0x%x)\n", _task_get_error() );

    // get qid for motion detect task
    motion_detect_qid = _msgq_get_id(Q67_1C_ID, MOTION_DETECT_2_Q);
    if (motion_detect_qid == MSGQ_NULL_QUEUE_ID)
        mbox_printf("Display_Driver_Task(): Failure in call to _msgq_get_id(). (Code 0x%x)\n",
             _task_get_error() );

    // get qid for galvo data notifier
    galvo_notifier_qid = _msgq_get_id(Q67_1B_ID, GALVO_NOTIFIER_SYS_Q);
    if ( galvo_notifier_qid == MSGQ_NULL_QUEUE_ID )
        mbox_printf("Display_Driver_Task(): Failure in call to _msgq_get_id(). (Code 0x%x)\n",
            _task_get_error() );

    // allocate memory for display_driver_struct
    display_driver_struct_ptr = _mem_alloc( sizeof(DISPLAY_DRIVER_DATA_STRUCT) );
    if (display_driver_struct_ptr == NULL)
        mbox_printf("Display_Driver_Task(): Failure in call to _mem_alloc(). (Code 0x%x\n",
            _task_get_error() );


    while( TRUE )
    {
        // allocate motion request message
loop0:
        rqst_msg_ptr = _msg_alloc( rqst_msg_pool_id );
        if (rqst_msg_ptr == NULL)
            mbox_printf("Display_Driver_Task(): Failure in call to _msg_alloc() <motion>.
                (code 0x%x)\n", _task_get_error() );

        // build rqst message for Motion_Detect_Task()
        rqst_msg_ptr->HEADER.SIZE = sizeof( UTILITY_MESSAGE );
        rqst_msg_ptr->HEADER.SOURCE_QID = display_driver_qid;
        rqst_msg_ptr->HEADER.TARGET_QID = motion_detect_qid;
        rqst_msg_ptr->DATA = RQST_NEXT_DATA;

        // send request message for most recent MOTION data
        if ( _msgq_send( (pointer)rqst_msg_ptr ) == FALSE )
            mbox_printf("Display_Driver_Task(): Failure in call to _msgq_send().
                (Code 0x%x)\n", _task_get_error() );

        // process MOTION detect message
        display_driver_msg_ptr = _msgq_receive( display_driver_qid, 0 );
        if (display_driver_msg_ptr->TYPE != MOTION)    // lost sync
        {   // discard and restart
            mbox_printf("Display_Driver_Task(): Lost Sync\n");
            _msg_free((pointer)display_driver_msg_ptr);
            goto loop0;
        } //Endif

        // copy MOTION message to struct
        count = display_driver_msg_ptr->DATA.COUNT;
        display_driver_struct_ptr->COUNT = count;
```

displaydriver.c continued

```
        for (i = 0; i < count; ++i )
        {
          display_driver_struct_ptr->INDEX[i] = display_driver_msg_ptr->DATA.INDEX[i];
          display_driver_struct_ptr->VALUE[i] = display_driver_msg_ptr->DATA.VALUE[i];
        } //Endfor

        _msg_free( (pointer)display_driver_msg_ptr );

        // allocate galvo position request message
        galvo_pos_rqst_msg_ptr = _msg_alloc(rqst_msg_pool_id);
        if (galvo_pos_rqst_msg_ptr == NULL)
            mbox_printf("Display_Driver_Task(): Failure in call to _msg_alloc() <galvo>.
                (Code 0x%x)\n", _task_get_error() );

        // build request message for Galvo_Data_Notifier()
        galvo_pos_rqst_msg_ptr->HEADER.SIZE = sizeof( UTILITY_MESSAGE );
        galvo_pos_rqst_msg_ptr->HEADER.SOURCE_QID = display_driver_qid;
        galvo_pos_rqst_msg_ptr->HEADER.TARGET_QID = galvo_notifier_qid;
        galvo_pos_rqst_msg_ptr->DATA = RQST_NEXT_DATA;

        // send request message for most recent GALVO data
        if( _msgq_send( (pointer)galvo_pos_rqst_msg_ptr ) == FALSE )
            mbox_printf("Display_Driver_Task(): Failure in call to _msgq_send() <galvo>.
                (Code 0x%x)\n", _task_get_error() );

        // process GALVO detect message
        display_driver_msg_ptr = _msgq_receive( display_driver_qid, 0 );
        if (display_driver_msg_ptr->TYPE != GALVO) // lost sync
        {   // discard and restart
            mbox_printf("Display_Driver_Task(): Lost Sync\n");
            _msg_free((pointer)display_driver_msg_ptr);
            goto loop0;
        } //Endif

        // add GALVO posn info to struct
        // count is specified by motion detect task, not galvo
        // Note that the galvo_isr transfers the positions of ALL points.
        // However only positions associated with points in motion, i.e.
        // at display_driver_struct_ptr->INDEX[i] are required (e.g. 0,1,5,100,101 etc.).
        // A total of 'count' points should be transferred and displayed.
        for (i = 0; i < count; ++i )
        {
            display_driver_struct_ptr->X_GALVO_POS[i] =
                display_driver_msg_ptr->DATA.X_GALVO_POS[display_driver_struct_ptr->INDEX[i]];

            display_driver_struct_ptr->Y_GALVO_POS[i] =
                display_driver_msg_ptr->DATA.Y_GALVO_POS[display_driver_struct_ptr->INDEX[i]];

        } //Endfor

        _msg_free( (pointer)display_driver_msg_ptr );

        // transfer data to PC for display
        mbox_display_data( display_driver_struct_ptr );

    } //Endwhile

} //Endbody
```

mboxdisplaydat.c

```
/*
 *
 * mbox_display_data()
 *
 *  FILENAME:     mboxdisplaydat.c
 *
 *  PARAMETERS:pointer to display driver message (DISPLAY_DRIVER_MSG_PTR )
 *
 *  DESCRIPTION:
 *              writes a raw data to mailbox for transfer to host PC
 *
 *              When        Who   What
 *              2001-10-05  D.G   original version
 *              2001-11-21  DG    sends 0 length messages
 *              2001-11-30  DG    changed parameter declarations
 *
 *  RETURNS:
 *              number of points transferred.
 *
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *
 */

uint_32 mbox_display_data( DISPLAY_DRIVER_DATA_STRUCT_PTR display_driver_struct_ptr )
{

   uint_32          i;
   uint_32          count;

   count = display_driver_struct_ptr->COUNT;

   // check status using blocking delay to allow other tasks run
   while (check_outbox(0) == 0); // wait while full

   mailbox_interrupt( count ); // send message length (number of points to display)
   write_mailbox( GRAPHIC_POINT, 0 ); // send message TYPE (String, graphic)

   // transfer data for display
   if (count > 0)
   {
     for (i = 0; i < count; ++i )
      {
        write_mailbox( display_driver_struct_ptr->INDEX[i], 0 );
        write_mailbox( display_driver_struct_ptr->VALUE[i], 0 );
        write_mailbox( display_driver_struct_ptr->X_GALVO_POS[i], 0 );
        write_mailbox( display_driver_struct_ptr->Y_GALVO_POS[i], 0 );
      } //Endfor
   } //Endif

   return(count);
} //Endbody
```

user.c

```
/*****************************************************************
 *
 *   DESCRIPTION:
 *         User Interface Task for Motion Detector Demo
 *         using Quatro67.
 *
 *         Derived directly from WaveformGui\demo0
 *
 *         This demo runs with a GUI written in Visual Basic:
 *         ..\MotionDemos\demo0\VB\motion0.vbp
 *
 *         Direct access to Innovative's "terminal" applet is not
 *         supported. All stdio for cpu_a uses the mailbox.
 *         Other processors continue to use the stdio out box in CCS
 *         for debugging purposes only.
 *
 *         Waveform generation uses dual axis galvo control.
 *
 *         IMAGE MODE acquisitions are not supported.
 *         Line acquisitions are supported.
 *         FILE HANDLING is not supported.
 *
 *
 *   AUTHOR:     D. Green
 *
 *   HISTORY:        Original version derived 2001-04-25
 *         version 2001-05-11
 *         version 2001-05-17 2-axis, multiple waveforms
 *         version 2001-06-08 generalized waveforms
 *
 *         version 2001-07-11 rewritten for gui interface
 *         version 2001-08-27 changed for motion detection app.
 *         version 2001-11-08 added parameters for motion detection
 *
 *                         Waveform Generation Table
 *      ---------------------------------------------------------------------------------
 *      Waveform     x_Vox_inc     y_Vox_inc     x_raster_inc      y_raster_inc
 *      ---------------------------------------------------------------------------------
 *      Lissajous    x_cycles      y_cycles      0                 0
 *      Vector       1             1             0                 0
 *      Raster-x     1             0             0                 1
 *      Raster-y     0             1             1                 0
 *
 *      e.g. For a raster with x-axis the primary (fast) axis, use
 *      x_raster_inc = 0, y_raster_inc = 1
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *******************************************************************/

/*TASK*-----------------------------------------------------------------
*
* Task Name : User_Interface_Task
* Comments  :
*
*END*-----------------------------------------------------------------*/
```

user.c continued

```
void User_Interface_Task
    (
        uint_32 parameter
    )
/*
    User Interface Task
*/

{ /* Body */

    uint_32         in_command, edit_command;
    boolean         DONE, SCANNING, normal_stop;
    boolean         loop, first_pass;
    uint_32         result;
    pointer         *error_ptr;
    pointer         *msg_error_ptr;

    float const_    DEG_TO_RAD = 0.0174533;
    double          x_val, y_val;
    uint_16         wave_index;
    uint_32         x_wave_type, y_wave_type;
    float           x_amp, y_amp;
    float           x_offset, y_offset;
    float           x_phase, y_phase;
    float           x_phase_correction, y_phase_correction;
    uint_32         x_vox_per_line, y_vox_per_line;
    uint_16         x_cycles, y_cycles;
    uint_16         x_raster_inc, y_raster_inc;
    int_32          val;

    int_32          motion_direction;
    float           motion_filter_constant;
    int_32          motion_threshold;

    uint_32         i;
    uint_16         k;// new wave index

    _task_id        Galvo_control_id;     // Galvo_data_id;
    _task_id        Peak_data_id;
    _queue_id       galvo_control_qid;    //galvo_data_qid;
    _queue_id       user_qid, peak_qid;

    WAVEFORM_MASTER_STRUCT_PTR volatile         waveform_master_ptr;
    GALVO_CMD_MESSAGE_STRUCT_PTR volatile       galvo_cmd_msg_ptr;
    UTILITY_MESSAGE_PTR volatile                ack_msg_ptr, galvo_utility_msg_ptr;
    PEAK_CMD_MESSAGE_PTR volatile               peak_cmd_msg_ptr;

    // perform startup handshake with Host
    in_command = mbox_get();
    mbox_put(CMD_OK);     // all command handshakes are acknowledged (protocol)
    while (in_command != LAUNCH_TARGET);//busywait

    /* initialization */
    normal_stop = TRUE;
    DONE = FALSE;
    SCANNING = FALSE;

    // motion detection parameters
    // 2001-11-08 DG
    motion_direction = MOTION_DIR_BOTH;
    motion_filter_constant = MOTION_K_FACTOR;
    motion_threshold = MOTION_DIFF_THRESHOLD;
```

user.c continued

```
// build waveform master struct, initialized to zero
    waveform_master_ptr =
        (WAVEFORM_MASTER_STRUCT_PTR)_mem_alloc_zero(sizeof(WAVEFORM_MASTER_STRUCT));

    if (waveform_master_ptr == NULL)
      mbox_printf("User_interface_task(). Error in call to _mem_alloc_zero(). [code: 0x%x]\n",
            _task_get_error());

    // Default waveform parameters
    wave_index = 0;
    x_wave_type = COS;
    y_wave_type = COS;
    x_amp = 2.0;
    y_amp = .5;
    x_offset = 0.;
    y_offset = 0.;

    //phase in degrees, converted to radians before transfer to galvo task.
    x_phase = 0.;
    y_phase = 0.;
    x_phase_correction = 0.;
    y_phase_correction = 0.;

    x_vox_per_line = 512;  // must be multiple of 128
    y_vox_per_line = x_vox_per_line;
    x_cycles = 2;
    y_cycles = 3;
    x_raster_inc = 0;
    y_raster_inc = 0;

    // waveform[0] selected as active waveform by default
    waveform_master_ptr->WAVE_INDEX = k = wave_index;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.WAVE_TYPE = x_wave_type;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.AMP = x_amp;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.OFFSET = x_offset;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.PHASE = x_phase;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.PHASE_CORRECTION =
        x_phase_correction;

    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.VOXELS_PER_LINE = x_vox_per_line;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.CYCLES = x_cycles;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.RASTER_INC = x_raster_inc;

    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.WAVE_TYPE = y_wave_type;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.AMP = y_amp;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.OFFSET = y_offset;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.PHASE = y_phase;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.PHASE_CORRECTION =
        y_phase_correction;

    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.VOXELS_PER_LINE = y_vox_per_line;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.CYCLES = y_cycles;
    waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.RASTER_INC = y_raster_inc;

    // mark all other waveforms 'undefined'
    for (i = 1; i < MAX_NUM_WAVEFORMS; ++i)
    {
        waveform_master_ptr->WAVEFORM[i].SCAN_INFO_X.WAVE_TYPE = UNDEFINED;
        waveform_master_ptr->WAVEFORM[i].SCAN_INFO_Y.WAVE_TYPE = UNDEFINED;
    }


    // 1s startup delay to allow other processors to initialize.
    _time_delay( 1000 );
```

user.c continued

```
/* create message pool - for galvo command messages */
if ( _msgpool_create_system(
    (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT), MAX_BUFFERS, 0, 0 ) == FALSE )
{
    mbox_printf("User_interface_task: Call to _msgpool_create_system() failed.
        (error code 0x%x)\n", _task_get_error() );
    _mqx_exit(0);
} /* Endif */

/* create message pool - for Peak_Data_Task messages */
if ( _msgpool_create_system(
    (uint_16)sizeof(PEAK_CMD_MESSAGE), MAX_BUFFERS, 0, 0 ) == FALSE )
{
    mbox_printf("User_interface_task: Call to _msgpool_create_system() failed.
        (error code 0x%x)\n", _task_get_error() );
    _mqx_exit(0);
} /* Endif */

/* open queue */
if ( _msgq_open(USER_INTERFACE_Q, 0) == 0 )
{
    mbox_printf("User_interface_task: Call to _msgq_open() failed.
        (error code 0x%x)\n", _task_get_error() );
    _mqx_exit(0);
}

/* get qids */
user_qid = _msgq_get_id( Q67_1A_ID, USER_INTERFACE_Q );
galvo_control_qid = _msgq_get_id(Q67_1B_ID, GALVO_CONTROL_Q);
peak_qid = _msgq_get_id(Q67_1C_ID, PEAK_DATA_Q);
if ( user_qid == 0 || galvo_control_qid == 0 || peak_qid == 0 )
    mbox_printf("User_interface_task() error: _msgq_get_id() returns '0'.\n");

/* create and start up Peak_Data_Task */
Peak_data_id = _task_create( Q67_1C_ID, PEAK_DATA_TASK, 0 );
if ( Peak_data_id == MQX_NULL_TASK_ID )
    mbox_printf("User_interface_task: Call to _task_create(PEAK_DATA_TASK) failed.
        (error code 0x%x)\n", _task_get_error() );

/* create Galvo_Control_Task */
Galvo_control_id = _task_create( Q67_1B_ID, GALVO_CONTROL_TASK, 0 );
if ( Galvo_control_id == MQX_NULL_TASK_ID )
    mbox_printf("User_interface_task: Call to _task_create(GALVO_CONTROL_TASK) failed.
        (error code 0x%x)\n", _task_get_error());

mbox_printf("Initializing... Do not click 'Init' until Waveform controller is ready.\n");
_time_delay( 8000 ); // wait 8s for CCS to initialize

mbox_printf("About to initialize Galvo controller.\n");

// initialize Galvo
galvo_cmd_msg_ptr =
    (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
    (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
if(galvo_cmd_msg_ptr == NULL)
{
    mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
        (error code 0x%x)\n", _task_get_error());
    _mqx_exit(0);
}

/* prepare initialization message for GALVO_CONTROL_TASK */
galvo_cmd_msg_ptr->HEADER.SIZE = (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
```

user.c continued

```
galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;

    galvo_cmd_msg_ptr->GALVO_CMD = INIT_GALVO;
    if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
        mbox_printf("User_interface_task: call to _msgq_send() failed. (error code 0x%x)\n",
            _task_get_error());

    /* wait for acknowledgement from Galvo_control_task */
    ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
    if ( ack_msg_ptr == NULL )
        mbox_printf("User_interface_task: Call to _msgq_receive() failed.
            (error code 0x%x)\n", _task_get_error());

    if (ack_msg_ptr->DATA != GALVO_OK)
        mbox_printf("Warning: Galvo controller not responding.\n");

    _msg_free( (pointer)ack_msg_ptr);

    // define default waveform (0)
    galvo_cmd_msg_ptr =
        (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
        (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
    if(galvo_cmd_msg_ptr == NULL)
    {
        mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
            (error code 0x%x)\n", _task_get_error());
        _mqx_exit(0);
    }

    // prepare initialization message for GALVO_CONTROL_TASK
    galvo_cmd_msg_ptr->HEADER.SIZE = (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
    galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
    galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;

    galvo_cmd_msg_ptr->GALVO_CMD = DEFINE_WAVE;
    galvo_cmd_msg_ptr->WAVE_INDEX = wave_index;

    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.WAVE_TYPE = x_wave_type;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.AMP = x_amp;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.OFFSET = x_offset;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE = x_phase * _DEG_TO_RAD;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE_CORRECTION =
        x_phase_correction * _DEG_TO_RAD;

    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.VOXELS_PER_LINE = x_vox_per_line;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.CYCLES = x_cycles;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.RASTER_INC = x_raster_inc;

    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.WAVE_TYPE = y_wave_type;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.AMP = y_amp;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.OFFSET = y_offset;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE = y_phase * _DEG_TO_RAD;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE_CORRECTION =
        y_phase_correction * _DEG_TO_RAD;

    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.VOXELS_PER_LINE = y_vox_per_line;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.CYCLES = y_cycles;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.RASTER_INC = y_raster_inc;

    // send default waveform definition to the Galvo_Control_Task
    if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
        mbox_printf("User_interface_task: call to _msgq_send() failed.
            (error code 0x%x)\n", _task_get_error());
    // wait for acknowledgement from Galvo_Task
```

user.c continued

```
    ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );

    if ( ack_msg_ptr == NULL )
        mbox_printf("User_interface_task: Call to _msgq_receive() failed.
            (error code 0x%x)\n", _task_get_error());

    if (ack_msg_ptr->DATA != GALVO_OK)
        mbox_printf("Warning: Galvo controller not responding.\n");

    _msg_free( (pointer)ack_msg_ptr);
    if (_task_get_error() != 0)
        mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
            _task_get_error());

    mbox_printf("\nWaveform controller ready.\n");

    while( !DONE )
    {
        // check for stack overrun
        if ( _task_check_stack() )
        {
            mbox_printf("Fatal Error in User Interface Task. Stack overflow detected.\n
                Program is terminating.\n");
            _mqx_exit(0);
        }

        // test memory
        if (result = _mem_test() != MQX_OK)
            mbox_printf("User_interface_task() memory error [code 0x%x].\n", result);

        // test message pools
        if (result = _msgpool_test(error_ptr, msg_error_ptr) != MQX_OK)
        {
            mbox_printf("User_interface_task() msgpool error [code 0x%x].\n", result);
            mbox_printf("Bad Message pool: 0x%x\n", error_ptr);
            mbox_printf("Bad message: 0x%x\n", msg_error_ptr);
        }

        // test message queues
        if (result = _msgq_test( error_ptr, msg_error_ptr) != MQX_OK)
            mbox_printf("User_interface_task() msg_queue error [code 0x%x].\n", result);

        // check error codes.
        if ( _task_get_error() != 0 )
            mbox_printf("User_interface_task() error code 0x%x\n", _task_get_error());

        if ( normal_stop == TRUE )
            in_command = mbox_get();


/*  Original user commands:                                    Replaced by:
    ---------------------------                                -------------

    puts("1 - Display pinout info.\n");                        WVFRM_SHOW_PINS
    puts("2 - Update current waveform.\n");                    WVFRM_EDIT
    puts("3 - Define waveform.\n");                            WVFRM_DEFINE
    puts("4 - Select waveform.\n");                            WVFRM_SELECT
    puts("5 - Display waveform parameters (local).\n");        WVFRM_SHOW_PARAMS
//  puts("6 - Debug: display waveform parameters (remote).\n"); n/a
    puts("g - Start scanning.\n");                             WVFRM_START_SCANNING
    puts("s - Stop continuous scanning.\n");                   WVFRM_STOP_SCANNING
    puts("t - test communications with Galvo control task.\n"); n/a
    puts("q - Quit\n");                                        WVFRM_QUIT
*/
```

user.c continued

```
/*
    New Command for Motion Detection:
    --------------------------------
    WVFRM_MOTION_PARAMS
*/


        normal_stop = TRUE;

        switch( in_command )
        {
            case WVFRM_SHOW_PINS:  /* display pinout info */
            {
                mbox_printf("X Ramp output:\t\t\t\tA4D4 Pin 44\n");
                mbox_printf("Y Ramp output:\t\t\t\tA4D4 Pin 45\n");
                mbox_printf("Voxel_CLK input:\t\t\tGalvo VCLK Pin 1\n");
                mbox_printf("FIFO_EN output:\t\t\t\tGalvo DIO Pin 6\n");
                mbox_printf("HSYNC output: \t\t\t\tGalvo DIO Pin 4\n");
                mbox_printf("VSYNC output: \t\t\t\tGalvo DIO Pin 5\n");
                mbox_printf("Galvo busy:\t\t\t\t\tGalvo DIO Pin 7\n");
                mbox_printf("Peak detector busy:\t\t\tPeak DIO Pin 6\n\n");
                mbox_printf("CCD data is processed on-the-fly by the peak detector.\n");
                break;
            } // Endcase

            case WVFRM_MOTION_PARAMS:    // adjust parameters for motion detection
            {                            // fetch Direction, Filter_Constant, Threshold
                mbox_put(CMD_OK);
                mbox_scanf("%d", &motion_direction);
                mbox_scanf("%f", &x_val); // must use a double
                motion_filter_constant = (float)x_val;
                mbox_scanf("%d", &motion_threshold);
                break;
            } //Endcase

            case WVFRM_EDIT:  // edit current waveform
            {
                first_pass = TRUE;
                loop = TRUE;
                while ( loop )
                {
                    if (first_pass)
                    {
                        mbox_put(CMD_OK);
                        first_pass = FALSE;
                    }

                    edit_command = mbox_get();
                    switch( edit_command )
                    {
                        case WVFRM_EDIT_AMPLITUDE:
                        {

                            mbox_scanf("%f", &x_val);
                            mbox_scanf("%f", &y_val);
                            x_amp = (float)x_val;
                            y_amp = (float)y_val;

                            // update master struct
                            waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_X.AMP = x_amp;
                            waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_Y.AMP = y_amp;
```

user.c continued

```
    // prepare to send update to galvo control task
    galvo_cmd_msg_ptr =
        (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
        (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
    if(galvo_cmd_msg_ptr == NULL)
    {
        mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
            (error code 0x%x)\n", _task_get_error());
        _mqx_exit(0);
    }
    galvo_cmd_msg_ptr->HEADER.SIZE =
        (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
    galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
    galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;

    galvo_cmd_msg_ptr->GALVO_CMD = NEW_AMP;
    galvo_cmd_msg_ptr->WAVE_INDEX = wave_index;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.AMP = x_amp;
    galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.AMP = y_amp;

    if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
        mbox_printf("User_interface_task: call to _msgq_send() failed.
            (error code 0x%x)\n", _task_get_error());

    /* wait for acknowledgement from Galvo_control_task */
    ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
    if ( ack_msg_ptr == NULL )
        mbox_printf("User_interface_task: Call to _msgq_receive() failed.
            (error code 0x%x)\n", _task_get_error());

    if (ack_msg_ptr->DATA != GALVO_OK)
        mbox_printf("Warning: Galvo controller not responding.\n");

    _msg_free( (pointer)ack_msg_ptr);
    if (_task_get_error() != 0)
        mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
            _task_get_error());

    break;
} // Endcase

case WVFRM_EDIT_PHASE:
{
    mbox_scanf("%f", &x_val);
    mbox_scanf("%f", &y_val);
    while (x_val < 0.) // wrap -ve phase
        x_val = 360. + x_val;
    while (y_val < 0.)
        y_val = 360. + y_val;
    x_phase = (float)x_val;
    y_phase = (float)y_val;

    // update master struct
    waveform_master_ptr->
        WAVEFORM[wave_index].SCAN_INFO_X.PHASE = x_phase;
    waveform_master_ptr->
        WAVEFORM[wave_index].SCAN_INFO_Y.PHASE = y_phase;

    // prepare to send update to galvo control task
    galvo_cmd_msg_ptr =
        (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
        (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
    if(galvo_cmd_msg_ptr == NULL)
```

user.c continued

```
        {
            mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
                (error code 0x%x)\n", _task_get_error());
            _mqx_exit(0);
        }
        galvo_cmd_msg_ptr->HEADER.SIZE =
            (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
        galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
        galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;

        galvo_cmd_msg_ptr->GALVO_CMD = NEW_PHASE;
        galvo_cmd_msg_ptr->WAVE_INDEX = wave_index;

        // convert Phase to radians
        galvo_cmd_msg_ptr->
            WAVEFORM_DEF.SCAN_INFO_X.PHASE = x_phase * _DEG_TO_RAD;
        galvo_cmd_msg_ptr->
            WAVEFORM_DEF.SCAN_INFO_Y.PHASE = y_phase * _DEG_TO_RAD;

        if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
            mbox_printf("User_interface_task: call to _msgq_send() failed.
                (error code 0x%x)\n", _task_get_error());

        /* wait for acknowledgement from Galvo_control_task */
        ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
        if ( ack_msg_ptr == NULL )
            mbox_printf ("User_interface_task: Call to _msgq_receive() failed.
                (error code 0x%x)\n", _task_get_error());

        if (ack_msg_ptr->DATA != GALVO_OK)
            mbox_printf("Warning: Galvo controller not responding.\n");

        _msg_free( (pointer)ack_msg_ptr);
        if (_task_get_error() != 0)
            mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
                _task_get_error());
        break;
    } // Endcase

    case WVFRM_EDIT_OFFSET:
    {
        mbox_scanf("%f", &x_val);
        mbox_scanf("%f", &y_val);
        x_offset = (float)x_val;
        y_offset = (float)y_val;

        // update master struct
        waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_X.OFFSET = x_offset;
        waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_Y.OFFSET = y_offset;

        // prepare to send update to galvo control task
        galvo_cmd_msg_ptr =
            (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
            (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
        if(galvo_cmd_msg_ptr == NULL)
        {
            mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
                (error code 0x%x)\n", _task_get_error());
            _mqx_exit(0);
        }
        galvo_cmd_msg_ptr->HEADER.SIZE =
            (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
```

user.c continued

```
                galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
                galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;

                galvo_cmd_msg_ptr->GALVO_CMD = NEW_OFFSET;
                galvo_cmd_msg_ptr->WAVE_INDEX = wave_index;
                galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.OFFSET = x_offset;
                galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.OFFSET = y_offset;

                if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
                    mbox_printf("User_interface_task: call to _msgq_send() failed.
                        (error code 0x%x)\n", _task_get_error());

                /* wait for acknowledgement from Galvo_control_task */
                ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
                if ( ack_msg_ptr == NULL )
                    mbox_printf("User_interface_task: Call to _msgq_receive() failed.
                        (error code 0x%x)\n", _task_get_error());

                if (ack_msg_ptr->DATA != GALVO_OK)
                    mbox_printf("Warning: Galvo controller not responding.\n");

                _msg_free( (pointer)ack_msg_ptr);
                if (_task_get_error() != 0)
                    mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
                        _task_get_error());

                break;
            } // Endcase

            default:
                loop = FALSE;
                break;
        } // Endswitch
    } // Endwhile

    break;
} // Endcase

case WVFRM_DEFINE:
{
    if (SCANNING)
    {
        mbox_put(CMD_REJECTED);
        mbox_printf("Warning: current scan must be terminated before a
            definition can be applied.\07\n");
        break;
    }

    mbox_put(CMD_OK);

    mbox_scanf("%d", &val); // needs an int_32
    k = (uint_16)val;
    waveform_master_ptr->WAVE_INDEX = k;

    // x waveform definition
    mbox_scanf("%d", &x_wave_type);

    if (x_wave_type == SAWTOOTH)
    {
        val = 2;
        while (val != 0 & val != 1)
        {
            mbox_scanf("%d", &val);
```

user.c continued

```
    } // Endwhile

    if (val == 0)
    {
        y_raster_inc = 1;
        x_raster_inc = 0;
    }

    else
    {
        x_raster_inc = 1;
        y_raster_inc = 0;
    }

} // Endif

mbox_scanf("%f", &x_val);
x_amp = (float)x_val;

mbox_scanf("%f", &x_val);
x_offset = (float)x_val;

mbox_scanf("%f", &x_val);
while (x_val < 0.) // wrap -ve phase
    x_val = 360. + x_val;
x_phase = (float)x_val;

mbox_scanf("%f", &x_val);
while (x_val < 0.) // wrap
    x_val = 360. + x_val;
x_phase_correction = (float)x_val;

mbox_scanf("%d", &x_vox_per_line);

mbox_scanf("%d", &val);
x_cycles = val;

// y waveform definition
mbox_scanf("%d", &y_wave_type);
mbox_scanf("%f", &y_val);
y_amp = (float)y_val;

mbox_scanf("%f", &y_val);
y_offset = (float)y_val;

mbox_scanf("%f", &y_val);
while (y_val < 0.) // wrap -ve phase
    y_val = 360. + y_val;
y_phase = (float)y_val;

mbox_scanf("%f", &y_val);
while (y_val < 0.) // wrap
    y_val = y_val + 360.;
y_phase_correction = (float)y_val;

mbox_scanf("%d", &y_vox_per_line);

mbox_scanf("%d", &val);
y_cycles = val;

// update master struct
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.WAVE_TYPE = x_wave_type;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.AMP = x_amp;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.OFFSET = x_offset;
```

user.c continued

```
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.PHASE = x_phase;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.PHASE_CORRECTION =
    x_phase_correction;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.VOXELS_PER_LINE =
    x_vox_per_line;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.CYCLES = x_cycles;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_X.RASTER_INC = x_raster_inc;

waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.WAVE_TYPE = y_wave_type;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.AMP = y_amp;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.OFFSET = y_offset;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.PHASE = y_phase;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.PHASE_CORRECTION =
    y_phase_correction;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.VOXELS_PER_LINE =
    y_vox_per_line;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.CYCLES = y_cycles;
waveform_master_ptr->WAVEFORM[k].SCAN_INFO_Y.RASTER_INC = y_raster_inc;

// prepare to send definition to galvo control task
galvo_cmd_msg_ptr =
    (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
    (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
if(galvo_cmd_msg_ptr == NULL)
{
    mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
        (error code 0x%x)\n", _task_get_error());
    _mqx_exit(0);
}

/* build message for GALVO_CONTROL_TASK */
galvo_cmd_msg_ptr->HEADER.SIZE =
    (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;

galvo_cmd_msg_ptr->GALVO_CMD = DEFINE_WAVE;
galvo_cmd_msg_ptr->WAVE_INDEX = k;

galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.WAVE_TYPE = x_wave_type;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.AMP = x_amp;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.OFFSET = x_offset;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE = x_phase * _DEG_TO_RAD;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE_CORRECTION =
    x_phase_correction * _DEG_TO_RAD;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.VOXELS_PER_LINE =
    x_vox_per_line;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.CYCLES = x_cycles;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.RASTER_INC = x_raster_inc;

galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.WAVE_TYPE = y_wave_type;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.AMP = y_amp;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.OFFSET = y_offset;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE = y_phase * _DEG_TO_RAD;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE_CORRECTION =
    y_phase_correction * _DEG_TO_RAD;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.VOXELS_PER_LINE =
    y_vox_per_line;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.CYCLES = y_cycles;
galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.RASTER_INC = y_raster_inc;
```

user.c continued

```
        /* send the update to the Galvo_Control_Task */
        if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
            mbox_printf("User_interface_task: call to _msgq_send() failed.
                (error code 0x%x)\n", _task_get_error());
        /* wait for acknowledgement from Galvo_Task */
        ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
        if ( ack_msg_ptr == NULL )
            mbox_printf("User_interface_task: Call to _msgq_receive() failed.
                (error code 0x%x)\n", _task_get_error());

        if (ack_msg_ptr->DATA != GALVO_OK)
            mbox_printf("Warning: Galvo controller not responding.\n");

        _msg_free( (pointer)ack_msg_ptr);
        if (_task_get_error() != 0)
            mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
                _task_get_error());

        break;
    } // Endcase

    case WVFRM_SELECT:    // select waveform
    {
        mbox_scanf("%d", &val);
        wave_index = (uint_16)val;

        // check if initialized
        if (waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_X.WAVE_TYPE == UNDEFINED
            || waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_Y.WAVE_TYPE
            == UNDEFINED)
        {
            mbox_printf("Waveform is undefined.\n");
        }

        // update master struct
        x_wave_type = waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_X.WAVE_TYPE;
        x_amp = waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_X.AMP;
        x_offset = waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_X.OFFSET;
        x_phase = waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_X.PHASE;
        x_phase_correction = waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_X.PHASE_CORRECTION;
        x_vox_per_line = waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_X.VOXELS_PER_LINE;
        x_cycles = waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_X.CYCLES;
        x_raster_inc = waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_X.RASTER_INC;

        y_wave_type = waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_Y.WAVE_TYPE;
        y_amp = waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_Y.AMP;
        y_offset = waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_Y.OFFSET;
        y_phase = waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_Y.PHASE;
        y_phase_correction = waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_Y.PHASE_CORRECTION;
        y_vox_per_line = waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_Y.VOXELS_PER_LINE;
        y_cycles = waveform_master_ptr->WAVEFORM[wave_index].SCAN_INFO_Y.CYCLES;
        y_raster_inc = waveform_master_ptr->
            WAVEFORM[wave_index].SCAN_INFO_Y.RASTER_INC;
```

user.c continued

```
    /* setup CHANGE_WAVEFORM message */
    galvo_cmd_msg_ptr = (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
        (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
    if(galvo_cmd_msg_ptr == NULL)
    {
        mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
            (error code 0x%x)\n", _task_get_error());
        _mqx_exit(0);
    }

    /* send */
    galvo_cmd_msg_ptr->HEADER.SIZE = sizeof(GALVO_CMD_MESSAGE_STRUCT);
    galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
    galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;
    galvo_cmd_msg_ptr->GALVO_CMD = CHANGE_WAVEFORM;
    galvo_cmd_msg_ptr->WAVE_INDEX = wave_index;

    if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
        mbox_printf("User_interface_task: call to _msgq_send() failed.
            (error code 0x%x)\n", _task_get_error());
    /* wait for acknowledgement from Galvo_Task */
    ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
    if ( ack_msg_ptr == NULL )
        mbox_printf("User_interface_task: Call to _msgq_receive() failed.
            (error code 0x%x)\n", _task_get_error());

    if (ack_msg_ptr->DATA != GALVO_OK)
        mbox_printf("Warning: Galvo controller not responding.\n");

    _msg_free( (pointer)ack_msg_ptr);
    if (_task_get_error() != 0)
        mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
            _task_get_error());
    break;
} // Endcase

case WVFRM_SHOW_PARAMS:  // display current waveform parameters
{
    mbox_put(CMD_OK);
    break; // Not implemented
} // Endcase

case '6':
{   // For testing: request a remote dump of current parameters
    mbox_put(CMD_OK);
    break; // not implemented
} // Endcase

case WVFRM_GO:
{
    if ( SCANNING )
    {
        mbox_put(CMD_REJECTED);
        mbox_printf("Warning: current scan must be terminated before
            starting a new scan.\07\n");
        break;
    } /* Endif */

    mbox_put(CMD_OK);
    SCANNING = TRUE;
```

user.c continued

```
/* send startup message to Peak_Data_Task */
peak_cmd_msg_ptr = (PEAK_CMD_MESSAGE_PTR)_msg_alloc_system(
    (uint_16)sizeof(PEAK_CMD_MESSAGE));
if ( peak_cmd_msg_ptr == NULL )
{
    mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
        (error code 0x%x\n", _task_get_error());
    _mqx_exit(0);
} /* Endif */

peak_cmd_msg_ptr->HEADER.SIZE = sizeof( PEAK_CMD_MESSAGE );
peak_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;
peak_cmd_msg_ptr->HEADER.TARGET_QID = peak_qid;
peak_cmd_msg_ptr->CMD = START_PEAK;
peak_cmd_msg_ptr->MOTION_DIRECTION = motion_direction;
peak_cmd_msg_ptr->MOTION_FILTER_CONSTANT = motion_filter_constant;
peak_cmd_msg_ptr->MOTION_DETECT_THRESHOLD = motion_threshold;

if (y_raster_inc != 0) // x is fast axis
    peak_cmd_msg_ptr->VOXELS_PER_LINE = x_vox_per_line;
else    // y is fast axis
    peak_cmd_msg_ptr->VOXELS_PER_LINE = y_vox_per_line;

if( _msgq_send( (pointer)peak_cmd_msg_ptr ) == 0 )
    mbox_printf("Interface_Task: Error in call to _msgq_send(). (Code 0x%x)\n",
        _task_get_error() );

/* wait for acknowledgement from Peak_Data_Task */
ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
if ( ack_msg_ptr == NULL )
    mbox_printf("User_interface_task: Call to _msgq_receive() failed.
        (error code 0x%x)\n", _task_get_error());
_msg_free( (pointer)ack_msg_ptr);
if (_task_get_error() != 0)
    mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
        _task_get_error());

/* setup START message */
galvo_cmd_msg_ptr = (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
    (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
if(galvo_cmd_msg_ptr == NULL)
{
    mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
        (error code 0x%x)\n", _task_get_error());
    _mqx_exit(0);
}

/* send the START command to galvo */
galvo_cmd_msg_ptr->HEADER.SIZE = sizeof(GALVO_CMD_MESSAGE_STRUCT);
galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;
galvo_cmd_msg_ptr->GALVO_CMD = START_GALVO;
galvo_cmd_msg_ptr->WAVE_INDEX = wave_index;

if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
    mbox_printf("User_interface_task: call to _msgq_send() failed.
        (error code 0x%x)\n", _task_get_error());

/* wait for acknowledgement from Galvo_Task */
ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
if ( ack_msg_ptr == NULL )
    mbox_printf("User_interface_task: Call to _msgq_receive() failed.
        (error code 0x%x)\n", _task_get_error());
```

user.c continued

```
        if (ack_msg_ptr->DATA != GALVO_OK)
            mbox_printf("Warning: Galvo controller not responding.\n");

        _msg_free( (pointer)ack_msg_ptr);
        if (_task_get_error() != 0)
            mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
                _task_get_error());

        break;
    }

    case WVFRM_STOP:  /* stop scanning */
    {
        /* stop galvo */
        SCANNING = FALSE;
        galvo_cmd_msg_ptr = (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
            (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
        if(galvo_cmd_msg_ptr == NULL)
            mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
                (error code 0x%x)\n", _task_get_error());

        /* prepare shutdown message for Galvo_Task */
        galvo_cmd_msg_ptr->HEADER.SIZE =
            (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
        galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
        galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;

        galvo_cmd_msg_ptr->GALVO_CMD = STOP_GALVO;

        /* stop the Galvo_Task */
        if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
            mbox_printf("User_interface_task: call to _msgq_send() failed.
                (error code 0x%x)\n", _task_get_error());

        /* wait for acknowledgement from Galvo_Task */
        ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
        if ( ack_msg_ptr == NULL )
            mbox_printf("User_interface_task: Call to _Receive_message() failed.
                (error code 0x%x)\n", _task_get_error());

        if (ack_msg_ptr->DATA != GALVO_OK)
            mbox_printf("Warning: Galvo controller not responding.\n");

        else
            mbox_printf("Galvo task has stopped.\n");

        _msg_free( (pointer)ack_msg_ptr);
        if (_task_get_error() != 0)
            mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
                _task_get_error());

        break;
    }

    case WVFRM_TEST_COMM:
    {   // test of communications with galvo_control_task()

        mbox_put(CMD_OK);
        galvo_cmd_msg_ptr = (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
            (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
        if(galvo_cmd_msg_ptr == NULL)
            mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
                (error code 0x%x)\n", _task_get_error());
```

user.c continued

```
        /* prepare message for Galvo_Task */
        galvo_cmd_msg_ptr->HEADER.SIZE =
            (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
        galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
        galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;
        galvo_cmd_msg_ptr->GALVO_CMD = TEST_CASE;

        if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
            mbox_printf("User_interface_task: call to _msgq_send() failed.
                (error code 0x%x)\n", _task_get_error());

        /* wait for acknowledgement from Galvo_Task */
        ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
        if ( ack_msg_ptr == NULL )
            mbox_printf("User_interface_task: Call to _Receive_message() failed.
                (error code 0x%x)\n", _task_get_error());

        if (ack_msg_ptr->DATA != GALVO_OK)
            mbox_printf("Warning: Galvo controller not responding.\n");

        else
            mbox_printf("Galvo communications are ok.\n");

        _msg_free( (pointer)ack_msg_ptr);
        if (_task_get_error() != 0)
            mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
                _task_get_error());

        break;
    }

    case WVFRM_QUIT:  /* quit */
    {
        /* stop galvo */
        SCANNING = FALSE;
        mbox_put(CMD_OK);
        galvo_cmd_msg_ptr = (GALVO_CMD_MESSAGE_STRUCT_PTR)_msg_alloc_system(
            (uint_16)sizeof(GALVO_CMD_MESSAGE_STRUCT));
        if(galvo_cmd_msg_ptr == NULL)
            mbox_printf("User_interface_task: Call to _msg_alloc_system() failed.
                (error code 0x%x)\n", _task_get_error());

        /* prepare shutdown message for Galvo_Task */
        galvo_cmd_msg_ptr->HEADER.SIZE =
            (_msg_size)sizeof( GALVO_CMD_MESSAGE_STRUCT );
        galvo_cmd_msg_ptr->HEADER.TARGET_QID = galvo_control_qid;
        galvo_cmd_msg_ptr->HEADER.SOURCE_QID = user_qid;

        galvo_cmd_msg_ptr->GALVO_CMD = STOP_GALVO;

        /* stop the Galvo_Task */
        if ( _msgq_send( (pointer)galvo_cmd_msg_ptr ) == FALSE )
            mbox_printf("User_interface_task: call to _msgq_send() failed.
                (error code 0x%x)\n", _task_get_error());

        /* wait for acknowledgement from Galvo_Task */
        ack_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
        if ( ack_msg_ptr == NULL )
            mbox_printf("User_interface_task: Call to _Receive_message() failed.
                (error code 0x%x)\n", _task_get_error());
```

user.c continued

```
            _msg_free( (pointer)ack_msg_ptr);
            if (_task_get_error() != 0)
                mbox_printf("User_interface_task(): error after _msg_free(), 0x%x\n",
                    _task_get_error());

            if ( _task_destroy( Galvo_control_id ) != MQX_OK )
                mbox_printf("User_interface_task: Call to _task_destroy() failed.
                    (error code 0x%x)\n", _task_get_error() );

            DONE = TRUE;
            break;
        }

        default:
        {
            mbox_put(CMD_UNKNOWN);
            mbox_printf("Unknown command <code 0x%x>\n", in_command);
        }
    } /* Endswitch */

}/* Endwhile */

/* destroy all tasks */
_mqx_exit(0);

} /* Endbody */
```

code67B.c

```
/* IIQ67 includes                          */
# include "c:\q6x\include\target\periph.h"
# include "c:\q6x\include\target\stdio.h"

/* MQX includes */
# include "c:\precise\mqx2.40\lib\3206701b.ti\mqx.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\fio.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\message.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\32060.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\mqx_prv.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\ipc.h"
# include "c:\precise\mqx2.40\lib\3206701b.ti\ipc_prv.h"
# include "c:\precise\mqx2.40\lib\iiq67b.ti\ipc_prv2.h"
# include "c:\precise\mqx2.40\lib\iiq67b.ti\ipcfifo1.h"
# include "c:\precise\mqx2.40\lib\iiq67b.ti\bsp.h"


/* application includes */
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\demodefs.h"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\demotyps.h"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\hardware.h"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\externsb.h"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\demo0B.c"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\galvonot.c"
# include "c:\Working\C6xScanner\mqx2.40\MotionDemosB\demo0\iiq67\galvo.c"

/*-----------------------------------------------------------------------
* (C), Her Majesty the Queen in right of Canada
* as represented by the National Research Council, Canada, 2001
-----------------------------------------------------------------------*/
```

externsb.h

```
/*****************************************************************
 *
 *    DESCRIPTION:   externsb.h
 *                   External declarations for Galvo_Control_Task and
 *                   Galvo_Data_Notifier.
 *
 *    AUTHOR:        D.Green
 *
 *    HISTORY:       First Version 2001-12-05
 *
 *****************************************************************/

# ifndef __Galvo_Externs
# define __Galvo_Externs

    GALVO_INT_STRUCT_PTR            galvo_buffer_ptr;
    GALVO_DATA_INFO_STRUCT_PTR      info_ptr;

# endif

/*-----------------------------------------------------------------
*   (C), Her Majesty the Queen in right of Canada
*   as represented by the National Research Council, Canada, 2001
-----------------------------------------------------------------*/
```

demo0B.c

```
extern void Galvo_Control_Task(uint_32 parameter);

TASK_TEMPLATE_STRUCT  MQX_template_list[] =
{

  { IPC_TTN, _ipc_task, IPC_DEFAULT_STACK_SIZE,6L, "_ipc_task", MQX_AUTO_START_TASK, 0L, 0L},
  { GALVO_CONTROL_TASK, Galvo_Control_Task, 4000L, 8L, "Galvo_control_task",
        MQX_FLOATING_POINT_TASK,  0L, 0L},
  { 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L}
};

MQX_INITIALIZATION_STRUCT  MQX_init_struct =
{
  /* PROCESSOR_NUMBER */                  Q67_1B_ID,
  /* START_OF_KERNEL_MEMORY */            BSP_DEFAULT_START_OF_KERNEL_MEMORY,
  /* END_OF_KERNEL_MEMORY */              BSP_DEFAULT_END_OF_KERNEL_MEMORY,
  /* INTERRUPT_STACK_SIZE */              BSP_DEFAULT_INTERRUPT_STACK_SIZE,
  /* TASK_TEMPLATE_LIST */                (pointer)MQX_template_list,
  /* MQX_HARDWARE_INT_LEVEL_MAX */        BSP_DEFAULT_MQX_HARDWARE_INTERRUPT_LEVEL_MAX,
  /* MAX_MSGPOOLS */                      BSP_DEFAULT_MAX_MSGPOOLS,
  /* MAX_MSGQS */                         BSP_DEFAULT_MAX_MSGQS,
  /* IO_CHANNEL */                        BSP_DEFAULT_IO_CHANNEL,
  /* IO_OPEN_MODE */                      BSP_DEFAULT_IO_OPEN_MODE
};

/*
    To avoid confusion, processors are referred to as
    Q67_A_ID etc. Since MQX refers to proc1-4
    and Innovative Integration refers to proc0-3.
*/

/* Off processor message routing <min, max, queue> */
IPC_ROUTING_STRUCT _ipc_routing_table[] =
{
 {Q67_1A_ID, Q67_1A_ID, FLINK1_B_A_Q},     /* links from cpu_b to cpu_a */
 {Q67_1C_ID, Q67_1C_ID, FLINK1_B_C_Q},     /* links from cpu_b to cpu_c */
 {Q67_1D_ID, Q67_1D_ID, FLINK1_B_D_Q},     /* links from cpu_b to cpu_d */
 {0,0,0}
};

 IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_B_A =
 {
    /* IPC on ProcB uses FIFOLinkB_A to Q67 processor A */
    /* DMA_CHANNEL                 */      0,
    /* MQX_DESTINATION_CPU         */      Q67_1A_ID,
    /* IN_PACKET_MAX_SIZE          */      512,
    /* IN_BUFFERS_TO_ALLOCATE */           16,
    /* IN_BUFFERS_TO_GROW          */      4,
    /* IN_BUFFERS_MAX_ALLOCATE*/           28,
    /* DEADLOCK_MASTER             */      IPC_C6X_DEADLOCK_SLAVE
 };

 IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_B_C =
 {
    /* IPC on ProcA uses FIFOLinkA_C to Q67 processor C    */
    /* DMA_CHANNEL                 */      1,
    /* MQX_DESTINATION_CPU         */      Q67_1C_ID,
    /* IN_PACKET_MAX_SIZE          */      512,
    /* IN_BUFFERS_TO_ALLOCATE */           16,
    /* IN_BUFFERS_TO_GROW          */      4,
    /* IN_BUFFERS_MAX_ALLOCATE*/           28,
    /* DEADLOCK_MASTER             */      IPC_C6X_DEADLOCK_MASTER
 };
```

demo0B.c continued

```
IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_B_D =
 {
    /* IPC on ProcA uses FIFOLinkA_D to Q67 processor D    */
    /* DMA_CHANNEL                  */      2,
    /* MQX_DESTINATION_CPU          */      Q67_1D_ID,
    /* IN_PACKET_MAX_SIZE           */      512,
    /* IN_BUFFERS_TO_ALLOCATE       */      16,
    /* IN_BUFFERS_TO_GROW           */      4,
    /* IN_BUFFERS_MAX_ALLOCATE*/             28,
    /* DEADLOCK_MASTER              */      IPC_C6X_DEADLOCK_SLAVE

 };


IPC_PROTOCOL_INIT_STRUCT _ipc_init_table[] =
{
    {   (uint_32 (_CODE_PTR_)(IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))
            _IPC_FIFOLINK_channel_init, (pointer)&IPC_FIFO_1D_init_rec_B_A,
            "FIFOLink_channel_A", FLINK1_B_A_Q},
    {   (uint_32 (_CODE_PTR_) (IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))
            _IPC_FIFOLINK_channel_init, (pointer)&IPC_FIFO_1D_init_rec_B_C,
            "FIFOLink_channel_C", FLINK1_B_C_Q},
    {   (uint_32 (_CODE_PTR_) (IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))
        _IPC_FIFOLINK_channel_init, (pointer)&IPC_FIFO_1D_init_rec_B_D,
            "FIFOLink_channel_D", FLINK1_B_D_Q},
    {   NULL, NULL, NULL, 0}
};

/*-----------------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 1998-2001
*-----------------------------------------------------------------------------*/

/* EOF */
```

galvonot.c

```
/*FUNCTION*-----------------------------------------------------------------
*
* Function Name   : Galvo_Data_Notifier
* Returned Value  : none
* Comments        :
*   This function handles the FIFOPort Almost Full interrupts (Half Full condition)
*   Arriving data is transferred by DMA to a data buffer managed by
*   the Galvo_Control_Task.
*
*   The notifier also polls for incoming requests for position data from the
*   Display_driver_task. Requests are acknowledged by a message containing a
*   GALVO_POSN_DATA struct.
*
* History
*       First Version 2001-12-04
*       Revision Date Who       What
*
*END*-----------------------------------------------------------------------*/

/*-----------------------------------------------------------------------
*   (C), Her Majesty the Queen in right of Canada
*   as represented by the National Research Council, Canada, 2001
-----------------------------------------------------------------------*/

void Galvo_Data_Notifier
  (
        /* [IN] the address of a GALVO_DATA_INFO_STRUCT_PTR */
        GALVO_DATA_INFO_STRUCT_PTR info_ptr
  )
{ /* Body */

    _queue_id                           Galvo_notifier_sys_qid;
    UTILITY_MESSAGE_PTR                 display_driver_rqst_msg_ptr;
    extern GALVO_INT_STRUCT_PTR         galvo_buffer_ptr;
    int_16                              * start_addr;
    DISPLAY_DRIVER_MSG_PTR              galvo_data_msg_ptr;

    uint_32                             line_index, point_index;
    uint_32                             i;

    // DMA from FIFOPort to the data buffer
    // [2001-03-22, DG]
    // FIFO is [512 x 16], so half buffer transfer is 256 x 16 or
    // 128 x/y galvo position pairs (each is an int_16)
    line_index = info_ptr->LINE_INDEX;
    point_index = info_ptr->POINT_INDEX;
    galvo_buffer_ptr = info_ptr->GALVO_DATA_PTR;
    start_addr = (int_16*)&(galvo_buffer_ptr->GALVO_DATA_ARRAY[line_index][point_index].BOTH);
    dma16_from_fifo
    (
        DMA_CH_3,                       // DMA channel
        (short*)&Periph->Fport + 1,     // SRC
        (short*)start_addr,             // dest
        (int)FIFOPORT_BUFFER_SIZE/2,    // count, 256 transfers = 128 voxels
        BLOCK                           // wait for DMA completion
    );

        // update contents of GALVO_DATA_INFO_STRUCT
        info_ptr->POINT_INDEX += SAMPLES_PER_PACKET;

        if ( info_ptr->POINT_INDEX >= info_ptr->VOXELS_PER_LINE ) // end of line
        {
            info_ptr->POINT_INDEX = 0;
```

galvonot.c continued

```
            info_ptr->LINE_INDEX += 1;
            if (info_ptr->LINE_INDEX >= MAX_BUFFERED_LINES) //wrap line index
                info_ptr->LINE_INDEX = 0;

            // check for incoming request message from Display_Driver_Task()
            Galvo_notifier_sys_qid = _msgq_get_id( On_This_Processor, GALVO_NOTIFIER_SYS_Q);
            display_driver_rqst_msg_ptr = _msgq_poll( Galvo_notifier_sys_qid );
            if (display_driver_rqst_msg_ptr != NULL)
            {
                // *** Handle the Request ***

                // reply to Display_Driver_Task() with display driver message.
                galvo_data_msg_ptr = _msg_alloc_system( sizeof(DISPLAY_DRIVER_MSG) );
                galvo_data_msg_ptr->HEADER.SIZE = sizeof(DISPLAY_DRIVER_MSG);
                galvo_data_msg_ptr->HEADER.SOURCE_QID = Galvo_notifier_sys_qid;
                galvo_data_msg_ptr->HEADER.TARGET_QID =
                    display_driver_rqst_msg_ptr->HEADER.SOURCE_QID;
                galvo_data_msg_ptr->TYPE = GALVO;
                galvo_data_msg_ptr->DATA.COUNT = 0; // this will be ignored by the correspondent task

                // copy most recent pos data to message
                for (i = 0; i < info_ptr->VOXELS_PER_LINE; ++i )
                {
                    galvo_data_msg_ptr->DATA.X_GALVO_POS[i] = galvo_buffer_ptr->
                        GALVO_DATA_ARRAY[line_index][i].PAIR.X_GALVO_POSN;
                    galvo_data_msg_ptr->DATA.Y_GALVO_POS[i] = galvo_buffer_ptr->
                        GALVO_DATA_ARRAY[line_index][i].PAIR.Y_GALVO_POSN;
                } //Endfor
                _msgq_send((pointer)galvo_data_msg_ptr);
                _msg_free((pointer)display_driver_rqst_msg_ptr);

            } //Endif

        } //Endif

    // clear the interrupt
    ClearInterrupt((int)EINT3_INTERRUPT);

} /* Endbody */
```

galvo.c

```
/*****************************************************************
 *
 *   DESCRIPTION:   Galvo Control Task
 *
 *                  This task is started and initialized by the User
 *                  Interface task. This version of the task
 *                  handles 3 message types:
 *                      - Start scan
 *                      - Stop scan
 *                      - Update scan parameters
 *
 *                  This task receives galvo requests from the user
 *                  interface task and passes them along to the galvo
 *                  processor which runs as a stand-alone application.
 *                  Communications between this task and the galvo processor
 *                  uses a FIFOPort driven by low-level PIO and DMA functions.
 *
 *                  Commands are issued to the galvo processor. No
 *                  acknowledgements are returned to this task.
 *
 *                  Version 2 installs the Galvo_data_notifier() and
 *                  initializes interrupts. Incoming FIFOPort data contains
 *                  galvo position data. No command acknowledgements are used.
 *
 *                  All stdout messages appear on a CCS display window.
 *
 *
 *   AUTHOR:        D. Green
 *
 *   HISTORY:       first version 2001-02-08
 *                  When        Who     What
 *                  2001-02-08  DG      first version
 *                  2001-12-04  DG      added Galvo_Data_Notifier
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *****************************************************************/



void Galvo_Control_Task
    (
        uint_32 parameter
    )

{ /* Body */

    uint_32 volatile            vector;
    uint_32 volatile            result;
    uint_32 volatile            response;
    boolean                     galvo_running;

    uint_32 volatile            start_en, end_en;
    pointer                     pool_err_ptr, msg_err_ptr;
    _queue_id                   galvo_control_qid;
    _queue_id                   galvo_notifier_qid;
    uint_16                     wave_index;
    uint_32                     voxels_per_line;

    int_32                      * x_ptr;
    int_32                      * y_ptr;
```

galvo.c continued

```
UTILITY_MESSAGE_PTR                     ack_msg_ptr;
GALVO_CMD_MESSAGE_STRUCT_PTR            galvo_cmd_msg_ptr;

extern GALVO_DATA_INFO_STRUCT_PTR       info_ptr;
extern GALVO_INT_STRUCT_PTR             galvo_buffer_ptr;

GALVO_WAVEFORM_TRANSFER_STRUCT_PTR      galvo_waveform_transfer_ptr;

puts("Galvo ok.\n"); // note: Stdout messages appear on CCS display window.

// initialize FIFOPort
enable_fifo_port();
reset_fifo_port();

// my AF threshold is set by remote processor via handshake
fifo_port_spit(INIT_FPORT);

// Galvo control processor uses a 1ms delay to read the fifoport
// wait here for an ack before proceeding
result = fifo_port_key();
if (result != PEAK_READY)
    _io_printf("Galvo_Control_Task(): Bad fifoport ack from Galvo Controller.\n");

/* open queue */
galvo_control_qid = _msgq_open( GALVO_CONTROL_Q, 0 );
if ( galvo_control_qid == 0 )
    _io_printf("Galvo_Control_Task: Error in call to _msgq_open().Code 0x%x\n", _task_get_error() );

// open system queue for galvo data notifier
galvo_notifier_qid = _msgq_open_system( GALVO_NOTIFIER_SYS_Q, 0, NULL, 0 );
if (galvo_notifier_qid == 0)
    _io_printf("Galvo_Control_Task: Error in call to _msgq_open_system().
        (Code 0x%x)\n", _task_get_error() );

/* create message pools */
if( _msgpool_create_system( (uint_16)sizeof( UTILITY_MESSAGE ), 16L, 0L, 0L ) == FALSE )
    _io_printf("Galvo_Control_Task: _msgpool_create_system() failed. (error code 0x%x)\n",
        _task_get_error() );

if ( _msgpool_create_system( sizeof(DISPLAY_DRIVER_MSG), 16L, 0L, 0L ) == FALSE )
    _io_printf("Galvo_Control_Task: _msgpool_create_system() failed. (error code 0x%x)\n",
        _task_get_error() );

if ( _msgpool_test( &pool_err_ptr, &msg_err_ptr ) != MQX_OK )
    _io_printf( "Galvo_Control_Task: bad message pool 0x%x or bad message 0x%x\n",
        pool_err_ptr, msg_err_ptr );

galvo_waveform_transfer_ptr = (GALVO_WAVEFORM_TRANSFER_STRUCT_PTR)
    _mem_alloc( sizeof(GALVO_WAVEFORM_TRANSFER_STRUCT) );
if (galvo_waveform_transfer_ptr == NULL)
    _io_printf("Galvo_Control_Task: _mem_alloc() failed. (error code 0x%x)\n", _task_get_error() );

// allocate buffer for galvo position data, "MAX_BUFFERED_LINES" are buffered.
// note that ALL galvo positions are transferred. The Display_Driver_Task will
// sort out which need be displayed based on motion detector data.
galvo_buffer_ptr = (GALVO_INT_STRUCT_PTR)_mem_alloc_system( sizeof(GALVO_INT_STRUCT) );
if (galvo_buffer_ptr == NULL)
{
    _io_printf("Galvo_Data_Task(): _mem_alloc_system(galvo_buffer) failed. (code 0x%x)\n",
        _task_get_error() );
    puts("Fatal error, program is terminating.\n");
    _mqx_exit(0);
}
```

galvo.c continued

```
// allocate galvo info struct
info_ptr =
    (GALVO_DATA_INFO_STRUCT_PTR)_mem_alloc_system( sizeof(GALVO_DATA_INFO_STRUCT) );
if (info_ptr == NULL)
{
    _io_printf("Galvo_Data_Task: _mem_alloc_system() failed. (error code 0x%x)\n",
        _task_get_error() );
    puts("Fatal error, program is terminating.\n");
    _mqx_exit(0);
}

// initialize galvo info struct

voxels_per_line = 0;
info_ptr->VOXELS_PER_LINE = voxels_per_line;
info_ptr->GALVO_DATA_PTR = galvo_buffer_ptr;
info_ptr->POINT_INDEX = 0;
info_ptr->LINE_INDEX = 0;

// Initialize FIFOPort Interrupt Handling
// Assign FPort Rx_AF interrupt (0x2) to portable int. code 3
InterruptSource((int)EINT3_INTERRUPT, (int)2);

// install Galvo_Data_Notifier()
vector = EINT3_INTERRUPT + 4; // offset by 4 from portable int code.
_int_install_isr( vector, (void (_CODE_PTR_)(pointer))Galvo_Data_Notifier, (pointer)info_ptr);

// use no interrupt polarity inversion for fifoport operation
INTR_EXT_POLARITY(3,0); // EXT_INT7
ClearInterrupt((int)EINT3_INTERRUPT);
EnableInterrupt((int)EINT3_INTERRUPT);

galvo_running = FALSE;

while ( TRUE )
{
    /* get a message buffer */
    ack_msg_ptr =
            (UTILITY_MESSAGE_PTR)_msg_alloc_system( (uint_16)sizeof(UTILITY_MESSAGE) );
    if( ack_msg_ptr == NULL )
        _io_printf("Galvo_Control_Task: Call to _msg_alloc_system() failed. (Error code 0x%x).\n",
            _task_get_error() );

    /* wait for next message */
    galvo_cmd_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
    if( galvo_cmd_msg_ptr == NULL )
        _io_printf("Galvo_Control_Task: Call to _msgq_receive() failed. (code 0x%x\n",
            _task_get_error() );
    wave_index = galvo_cmd_msg_ptr->WAVE_INDEX;

    switch (galvo_cmd_msg_ptr->GALVO_CMD)
    {
        case INIT_GALVO:
        {
            // initialize galvo to "stopped" state
            if ( (get_fifo_port_status() & Tx_FIFO_EMPTY) )
                fifo_port_spit( INIT_GALVO );
            else
            {
                ack_msg_ptr->DATA = GALVO_NOT_READY;
                break;
            }
            ack_msg_ptr->DATA = GALVO_OK;
            galvo_running = FALSE;
```

galvo.c continued

```
        // initialize galvo info struct
        info_ptr->GALVO_DATA_PTR = galvo_buffer_ptr;
        info_ptr->POINT_INDEX = 0;
        info_ptr->LINE_INDEX = 0;
        info_ptr->VOXELS_PER_LINE = voxels_per_line;
        break;
    } // Endcase

    case STOP_GALVO:
    {
        // stops galvo at the end of the current scan line.
        if ( (get_fifo_port_status() & Tx_FIFO_EMPTY) )
            fifo_port_spit( STOP_GALVO );
        else
        {
            ack_msg_ptr->DATA = GALVO_NOT_READY;
            break;
        }
        ack_msg_ptr->DATA = GALVO_OK;
        galvo_running = FALSE;
        break;
    } /* Endcase */

    case START_GALVO:
    {
        if ( galvo_running == FALSE )       // prevents false restarts
        {
            if ( (get_fifo_port_status() & Tx_FIFO_EMPTY) )
            {
                fifo_port_spit( START_GALVO );     // (no busywait)
                fifo_port_emit(wave_index);        // (busywait)
                ack_msg_ptr->DATA = GALVO_OK;
                galvo_running = TRUE;

                // initialize galvo info struct
                info_ptr->GALVO_DATA_PTR = galvo_buffer_ptr;
                info_ptr->POINT_INDEX = 0;
                info_ptr->LINE_INDEX = 0;
                info_ptr->VOXELS_PER_LINE = voxels_per_line;

                // flush incoming FIFOPort
                while (!(get_fifo_port_status() & Rx_FIFO_EMPTY))
                    fifo_port_eat();

                break;
            }

            ack_msg_ptr->DATA = GALVO_NOT_READY;
            break;
        } /* Endif */

        else
            ack_msg_ptr->DATA = GALVO_OK;     // already running, redundant request

        break;
    } /* Endcase */

    case CHANGE_WAVEFORM:
    {
        if ( (get_fifo_port_status() & Tx_FIFO_EMPTY) )
        {
            fifo_port_spit( CHANGE_WAVEFORM );
            fifo_port_emit(wave_index);
```

galvo.c continued

```
            ack_msg_ptr->DATA = GALVO_OK;
            break;
        }

        ack_msg_ptr->DATA = GALVO_NOT_READY;
        break;
    } // Endcase

    case DEFINE_WAVE:
    {
        if (!(get_fifo_port_status() & Tx_FIFO_EMPTY))
        {
            ack_msg_ptr->DATA = GALVO_NOT_READY;
            break;
        }

        voxels_per_line = galvo_cmd_msg_ptr->
            WAVEFORM_DEF.SCAN_INFO_X.VOXELS_PER_LINE;

        // DMA the entire struct: cmd, index, transfer size and parameters
        galvo_waveform_transfer_ptr->GALVO_CMD = DEFINE_WAVE;
        galvo_waveform_transfer_ptr->WAVE_INDEX = wave_index;
        galvo_waveform_transfer_ptr->TRANSFER_SIZE =
            (uint_16)sizeof( GALVO_WAVEFORM_TRANSFER_STRUCT ) / 2;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_X.WAVE_TYPE =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.WAVE_TYPE;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_X.AMP =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.AMP;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_X.OFFSET =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.OFFSET;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE_CORRECTION =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE_CORRECTION;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_X.VOXELS_PER_LINE =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.VOXELS_PER_LINE;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_X.CYCLES =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.CYCLES;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_X.RASTER_INC =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.RASTER_INC;


        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_Y.WAVE_TYPE =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.WAVE_TYPE;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_Y.AMP =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.AMP;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_Y.OFFSET =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.OFFSET;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE_CORRECTION =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE_CORRECTION;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_Y.VOXELS_PER_LINE =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.VOXELS_PER_LINE;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_Y.CYCLES =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.CYCLES;
        galvo_waveform_transfer_ptr->WAVEFORM_DEF.SCAN_INFO_Y.RASTER_INC =
            galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.RASTER_INC;
```

galvo.c continued

```
        dma16_to_fifo(
            DMA_CH_3,                                           // DMA channel # 3
            (short*)galvo_waveform_transfer_ptr,                // Source
            (short*)&Periph->Fport + 1,                         // Destination
            (int)sizeof(GALVO_WAVEFORM_TRANSFER_STRUCT)/2,      // DMA count
            BLOCK );                                            // wait

        ack_msg_ptr->DATA = GALVO_OK;
        break;
    } /* Endcase */

    case NEW_AMP:
    {   // update current waveform amplitudes
        if ( (get_fifo_port_status() & Tx_FIFO_EMPTY) )
        {
            // coerce the floats by using pointers
            x_ptr = (int_32 *)&galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.AMP;
            y_ptr = (int_32 *)&galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.AMP;
            fifo_port_spit( NEW_AMP );
            fifo_port_spit32((int)*x_ptr);
            fifo_port_spit32((int)*y_ptr);
            ack_msg_ptr->DATA = GALVO_OK;
            break;
        }

        ack_msg_ptr->DATA = GALVO_NOT_READY;
        break;
    } // Endcase

    case NEW_PHASE:
    {   // update current waveform phases
        if ( (get_fifo_port_status() & Tx_FIFO_EMPTY) )
        {
            x_ptr = (int_32 *)&galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.PHASE;
            y_ptr = (int_32 *)&galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.PHASE;
            fifo_port_spit( NEW_PHASE );
            fifo_port_spit32((int)*x_ptr);
            fifo_port_spit32((int)*y_ptr);
            ack_msg_ptr->DATA = GALVO_OK;
            break;
        }

        ack_msg_ptr->DATA = GALVO_NOT_READY;
        break;
    } // Endcase

    case NEW_OFFSET:
    {   // update current waveform phases
        if ( (get_fifo_port_status() & Tx_FIFO_EMPTY) )
        {
            x_ptr = (int_32 *)&galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_X.OFFSET;
            y_ptr = (int_32 *)&galvo_cmd_msg_ptr->WAVEFORM_DEF.SCAN_INFO_Y.OFFSET;
            fifo_port_spit( NEW_OFFSET );
            fifo_port_spit32((int)*x_ptr);
            fifo_port_spit32((int)*y_ptr);
            ack_msg_ptr->DATA = GALVO_OK;
            break;
        }

        ack_msg_ptr->DATA = GALVO_NOT_READY;
        break;
    } // Endcase
```

galvo.c continued

```
            case PRINT_VALUES:
            {
                if ( (get_fifo_port_status() & Tx_FIFO_EMPTY) )
                    fifo_port_spit( PRINT_VALUES );
                else
                {
                    ack_msg_ptr->DATA = GALVO_NOT_READY;
                    break;
                }
                ack_msg_ptr->DATA = GALVO_OK;
                break;
            }

            default:
            case TEST_CASE:
            {   // test of message passing
                ack_msg_ptr->DATA = GALVO_OK;
                break;
            }

        } /* Endswitch */

        /* reply to user interface task */
        ack_msg_ptr->HEADER.SIZE = sizeof( UTILITY_MESSAGE );
        ack_msg_ptr->HEADER.SOURCE_QID = galvo_control_qid;
        ack_msg_ptr->HEADER.TARGET_QID = galvo_cmd_msg_ptr->HEADER.SOURCE_QID;

        if( _msgq_send( (pointer)ack_msg_ptr ) == 0 )
            _io_printf("Galvo_Control_Task: Error in call to _msgq_send(). (Code 0x%x)\n",
                _task_get_error() );

        /* free incoming message buffer */
        _msg_free( (pointer)galvo_cmd_msg_ptr );
        if (_task_get_error() != 0)
            _io_printf("Galvo_control_task(): error after _msg_free(), 0x%x\n", _task_get_error());

    } /* Endwhile */

} /* Endbody */
```

code67c.c

```
/* IIQ67 includes                          */
#include "c:\q6x\include\target\periph.h"
#include "c:\q6x\include\target\stdio.h"

/* MQX includes */
#include "c:\precise\mqx2.40\lib\3206701b.ti\mqx.h"
#include "c:\precise\mqx2.40\lib\3206701b.ti\fio.h"
#include "c:\precise\mqx2.40\lib\3206701b.ti\message.h"
#include "c:\precise\mqx2.40\lib\3206701b.ti\32060.h"
#include "c:\precise\mqx2.40\lib\3206701b.ti\mqx_prv.h"
#include "c:\precise\mqx2.40\lib\3206701b.ti\ipc.h"
#include "c:\precise\mqx2.40\lib\3206701b.ti\ipc_prv.h"
#include "c:\precise\mqx2.40\lib\iiq67b.ti\ipc_prv2.h"
#include "c:\precise\mqx2.40\lib\iiq67b.ti\ipcfifo1.h"
#include "c:\precise\mqx2.40\lib\iiq67b.ti\bsp.h"


/* application includes */
#include "c:\Working\C6xScanner\mqx2.40\MotiondemosB\demo0\iiq67\demodefs.h"
#include "c:\Working\C6xScanner\mqx2.40\MotiondemosB\demo0\iiq67\demotyps.h"
#include "c:\Working\C6xScanner\mqx2.40\MotiondemosB\demo0\iiq67\hardware.h"
#include "c:\Working\C6xScanner\mqx2.40\MotiondemosB\demo0\iiq67\externs.h"
#include "c:\Working\C6xScanner\mqx2.40\MotiondemosB\demo0\iiq67\demo0C.c"
#include "c:\Working\C6xScanner\mqx2.40\MotiondemosB\demo0\iiq67\peaknot.c"
#include "c:\Working\C6xScanner\mqx2.40\MotiondemosB\demo0\iiq67\peakdata.c"
#include "c:\Working\C6xScanner\mqx2.40\MotiondemosB\demo0\iiq67\motiondetect.c"


/*-----------------------------------------------------------------------
*   (C), Her Majesty the Queen in right of Canada
*   as represented by the National Research Council, Canada, 2001
-----------------------------------------------------------------------*/
```

demo0c.c

```
//extern void Peak_Data_Task(uint_32 parameter);
//extern void Motion_Detect_Task(uint_32 parameter);

TASK_TEMPLATE_STRUCT  MQX_template_list[] =
{
  { IPC_TTN, _ipc_task, IPC_DEFAULT_STACK_SIZE, 6L, "_ipc_task", MQX_AUTO_START_TASK, 0L, 0L},
  { PEAK_DATA_TASK, Peak_Data_Task, 2500L, 7L, "peak_data_task", 0, 0L, 0L},
  { MOTION_DETECT_TASK, Motion_Detect_Task, 30000L, 8L, "motion_detect_task",
        MQX_FLOATING_POINT_TASK, 0L, 0L},
  { 0L, 0L, 0L, 0L, 0L, 0L, 0L, 0L}
};

MQX_INITIALIZATION_STRUCT  MQX_init_struct =
{
  /* PROCESSOR_ */                        Q67_1C_ID,
  /* START_OF_KERNEL_ */                  (pointer)0x02700000,
  /* END_OF_KERNEL_MEMORY */              (pointer)0x02ffffff,
  /* INTERRUPT_STACK_ */                  BSP_DEFAULT_INTERRUPT_STACK_SIZE,
  /* TASK_TEMPLATE_ */                    (pointer)MQX_template_list,
  /* MQX_HARDWARE_INT_LEVEL_MAX */        BSP_DEFAULT_MQX_HARDWARE_INTERRUPT_LEVEL_MAX,
  /* MAX_MSGPOOLS */                      BSP_DEFAULT_MAX_MSGPOOLS,
  /* MAX_MSGQS */                         BSP_DEFAULT_MAX_MSGQS,
  /* IO_CHANNEL */                        BSP_DEFAULT_IO_CHANNEL,
  /* IO_OPEN_MODE */                      BSP_DEFAULT_IO_OPEN_MODE
};

/*
    To avoid confusion, processors are referred to as
    Q67_A_ID etc. Since MQX refers to proc1-4
    and Innovative Integration refers to proc0-3.
*/

/* Off processor message routing <min, max, queue> */
IPC_ROUTING_STRUCT _ipc_routing_table[] =
{
 {Q67_1A_ID, Q67_1A_ID, FLINK1_C_A_Q},     /* links from cpu_c to cpu_a */
 {Q67_1B_ID, Q67_1B_ID, FLINK1_C_B_Q},     /* links from cpu_c to cpu_b */
 {Q67_1D_ID, Q67_1D_ID, FLINK1_C_D_Q},     /* links from cpu_c to cpu_d */
 {0,0,0}
};

 IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_C_A =
 {
    /* IPC on ProcC uses FIFOLinkC_A to Q67 processor A    */
    /* DMA_CHANNEL                 */      0,
    /* MQX_DESTINATION_CPU         */      Q67_1A_ID,
    /* IN_PACKET_MAX_SIZE          */      512,
    /* IN_BUFFERS_TO_ALLOCATE      */      16,
    /* IN_BUFFERS_TO_GROW          */      4,
    /* IN_BUFFERS_MAX_ALLOCATE     */      28,
    /* DEADLOCK_MASTER             */      IPC_C6X_DEADLOCK_SLAVE
 };

 IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_C_B =
 {
    /* IPC on ProcC uses FIFOLinkC_B to Q67 processor B    */
    /* DMA_CHANNEL                 */      1,
    /* MQX_DESTINATION_CPU         */      Q67_1B_ID,
    /* IN_PACKET_MAX_SIZE          */      512,
    /* IN_BUFFERS_TO_ALLOCATE      */      16,
    /* IN_BUFFERS_TO_GROW          */      4,
    /* IN_BUFFERS_MAX_ALLOCATE     */      28,
    /* DEADLOCK_MASTER             */      IPC_C6X_DEADLOCK_SLAVE
 };
```

demo0c.c continued

```
IPC_FIFO_1D_INIT_STRUCT IPC_FIFO_1D_init_rec_C_D =
 {
    /* IPC on ProcC uses FIFOLinkC_D to Q67 processor D   */
    /* DMA_CHANNEL                 */      2,
    /* MQX_DESTINATION_CPU         */      Q67_1D_ID,
    /* IN_PACKET_MAX_SIZE          */      512,
    /* IN_BUFFERS_TO_ALLOCATE      */      16,
    /* IN_BUFFERS_TO_GROW          */      4,
    /* IN_BUFFERS_MAX_ALLOCATE     */      28,
    /* DEADLOCK_MASTER             */      IPC_C6X_DEADLOCK_MASTER
 };


IPC_PROTOCOL_INIT_STRUCT _ipc_init_table[] =
{
    {   (uint_32 (_CODE_PTR_) (IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))
            _IPC_FIFOLINK_channel_init, (pointer)&IPC_FIFO_1D_init_rec_C_A,
            "FIFOLink_channel_A", FLINK1_C_A_Q},
    {   (uint_32 (_CODE_PTR_) (IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))
            _IPC_FIFOLINK_channel_init, (pointer)&IPC_FIFO_1D_init_rec_C_B,
            "FIFOLink_channel_B", FLINK1_C_B_Q},
    {   (uint_32 (_CODE_PTR_) (IPC_PROTOCOL_INIT_STRUCT_PTR, pointer))
            _IPC_FIFOLINK_channel_init, (pointer)&IPC_FIFO_1D_init_rec_C_D,
            "FIFOLink_channel_D", FLINK1_C_D_Q},
    {   NULL, NULL, NULL, 0}
};

/*-----------------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 1998-2001
*-----------------------------------------------------------------------------*/

/* EOF */
```

peaknot.c

```
/*FUNCTION*-------------------------------------------------------------
*
* Function Name   : Peak_Data_Notifier
* Returned Value  : none
* Comments        :
*   This function handles the FIFOPort Almost Full interrupts (Half Full condition)
*   Arriving data is transferred by DMA to a data buffer managed by
*   the Peak_Data_Task.
*   This code is designed to handle multiple peaks (colour systems).
*   The number of peaks is specified by NUMBER_OF_PEAKS in demodefs.h
*
* History
*       First Version 2001-03-xx
*       Revision Date     Who       What
*       2001-03-22        DG     changed to 16-bit data transfers from 32-bits.
*       2001-08-28        DG     added changes to support motion detection. Now handles
*                                line data rqst messages from motion detect task.
*       2001-11-16        DG     rearranged end-of-line and wrap detection code.
*
*END*-----------------------------------------------------------------*/

/*-----------------------------------------------------------------------
*   (C), Her Majesty the Queen in right of Canada
*   as represented by the National Research Council, Canada, 2001
-----------------------------------------------------------------------*/

void Peak_Data_Notifier
  (
    /* [IN] the address of a PEAK_DATA_INFO_STRUCT_PTR */
    PEAK_DATA_INFO_STRUCT_PTR info_ptr
  )
{ /* Body */

    _queue_id                   Peak_notifier_sys_qid;
    UTILITY_MESSAGE_PTR         motion_detect_rqst_msg_ptr;
    LINE_INFO_MSG_PTR           line_info_msg_ptr;

    // DMA from FIFOPort to the data buffer
    // [2001-03-22, DG]
    // FIFO is [512 x 16], so half buffer transfer is 256 x 16 half words or
    // no. of voxels transferred = 256 / (2 half-words/peak * no. peaks)
    dma16_from_fifo
    (
        DMA_CH_3,                           // DMA channel
        (short*)&Periph->Fport + 1,         // SRC
        (short*)info_ptr->BUFFER_PTR,       // dest
        (int)FIFOPORT_BUFFER_SIZE/2,        // count, 256 transfers = 128 voxels
        BLOCK                               // wait for DMA completion
    );

    // for correct startup synchronization:
    //   DO NOT discard data on a cold start
    //   DO discard data on a warm start.
    if (info_ptr->COLD_START)
    {
        info_ptr->COLD_START = FALSE;
        info_ptr->WARM_START = FALSE;
    }

    if (!info_ptr->WARM_START) // do not discard data
    {
        // update contents of PEAK_DATA_INFO_STRUCT
        info_ptr->BUFFER_PTR += FIFOPORT_BUFFER_SIZE/2; // no. uint_16s
```

peaknot.c continued

```
        info_ptr->TOTAL_VOXELS += (FIFOPORT_BUFFER_SIZE/2) / (2 * NUMBER_OF_PEAKS);
        info_ptr->VOXEL_CNTR += (FIFOPORT_BUFFER_SIZE/2) / (2 * NUMBER_OF_PEAKS);

        // Check for end of line
        if (info_ptr->VOXEL_CNTR >= info_ptr->VOXELS_PER_LINE)
        {   // End of Line
            info_ptr->VOXEL_CNTR = 0;
            info_ptr->LINE_NUM++;

            // check buffer limit, arrange to wrap on a full line
            if ( info_ptr->TOTAL_VOXELS >= (MAX_NUM_OF_VOXELS - info_ptr->VOXELS_PER_LINE) )
            {   // buffer overwrites on wrap
                // Image mode should never wrap, tracking mode will wrap.
                info_ptr->TOTAL_VOXELS = 0;
                info_ptr->BUFFER_PTR = info_ptr->REF_BUFFER_PTR;
                info_ptr->LINE_NUM = 0; // used as an index into the buffer
            } //Endif

            // check for incoming request message from Motion_Detect_Task()
            Peak_notifier_sys_qid = _msgq_get_id( On_This_Processor, PEAK_NOTIFIER_SYS_Q);
            motion_detect_rqst_msg_ptr = _msgq_poll( Peak_notifier_sys_qid );
            if (motion_detect_rqst_msg_ptr != NULL)
            {
                // *** Handle the Request ***

                // reply to Motion_Detect_Task() with line info message.
                line_info_msg_ptr = _msg_alloc_system( sizeof(LINE_INFO_MSG) );
                line_info_msg_ptr->HEADER.SIZE = sizeof(LINE_INFO_MSG);
                line_info_msg_ptr->HEADER.SOURCE_QID = Peak_notifier_sys_qid;
                line_info_msg_ptr->HEADER.TARGET_QID =
                    motion_detect_rqst_msg_ptr->HEADER.SOURCE_QID;
                line_info_msg_ptr->INFO_STRUCT_PTR = info_ptr;
                _msgq_send((pointer)line_info_msg_ptr);
                _msg_free((pointer)motion_detect_rqst_msg_ptr);

            } //Endif
        } //Endif

    }

    else // discard data on warm start
        info_ptr->WARM_START = FALSE;
    /* Endif */

    // clear the interrupt
    ClearInterrupt((int)EINT3_INTERRUPT);

} /* Endbody */
```

peakdata.c

```
/*****************************************************************
 *
 *   DESCRIPTION:  Peak Data Task
 *
 *                 This task installs the Peak_Data_Notifier which
 *                 handles FIFOPort interrupts from the Peak detector.
 *                 The task manages the incoming peak position and
 *                 intensity data. It builds a large data buffer for
 *                 this purpose.
 *
 *                 The task receives a START_PEAK command from the
 *                 User_Interface_Task which it uses to install or
 *                 initialize the notifier.
 *
 *                 All stdout messages appear on a CCS debugger output window.
 *
 *
 *   AUTHOR:       D.Green
 *
 *   HISTORY:      First Version 2001-02-16
 *                 Date          Who       What
 *                 2001-03-09    D.G.      added message transfers to file handler task
 *                 2001-08-28    DG        added creation of Motion_Detect_Task and
 *                                         PEAK_NOTIFIER_SYS_Q
 *                 2001-08-30    DG        added ACK to peak detector in START_PEAK case
 *                                         LINE_INFO_MSG msg pool.
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 ******************************************************************/


/*TASK*-----------------------------------------------------------------
*
* Task Name : Peak_Data_Task
* Comments  :
*
*END*-----------------------------------------------------------------*/


void Peak_Data_Task
   (
      uint_32 parameter
   )

{ /* Body */

   _queue_id                        peak_data_qid;
   _queue_id                        peak_notifier_sys_qid;
   _queue_id                        file_qid;

   _task_id                         Motion_detect_task_id;


   uint_16 volatile                 result;
   uint_32                          vector;
   uint_32                          line, voxels;
   uint_32 volatile                 first_point, last_point;
   uint_16                          *buffer_ptr;
   uint_32                          i;

   extern PEAK_DATA_BUFFER_PTR              peak_data_ptr;
   PEAK_DATA_BUFFER_PTR                     ref_peak_ptr;
```

peakdata.c continued

```
UTILITY_MESSAGE_PTR                     ack_msg_ptr;
PEAK_CMD_MESSAGE_PTR                    peak_cmd_msg_ptr;
PEAK_DATA_RPLY_MSG_PTR                  peak_data_rply_msg_ptr;
extern PEAK_DATA_INFO_STRUCT_PTR        info_ptr;



puts("Peak ok.\n"); // initial announcement

/* open queue */
peak_data_qid = _msgq_open( PEAK_DATA_Q, 0 );
if ( peak_data_qid == 0 )
    _io_printf("Peak_Data_Task: Error in call to _msgq_open(). Code 0x%x\n", _task_get_error() );

file_qid = _msgq_get_id( Q67_1A_ID, FILE_HANDLER_Q );
if (file_qid == MSGQ_NULL_QUEUE_ID)
    _io_puts("Peak_Data_Task: Error in call to _msgq_get_id(). Invalid processor number.");

/* create message pools */
if( _msgpool_create_system( (uint_16)sizeof( UTILITY_MESSAGE ), 16L, 0L, 0L ) == FALSE )
    _io_printf("Peak_Data_Task: _msgpool_create_system() failed. (error code 0x%x)\n",
        _task_get_error() );

if( _msgpool_create_system( (uint_16)sizeof( PEAK_DATA_RPLY_MSG ), 16L, 0L, 0L ) == FALSE )
    _io_printf("Peak_Data_Task: _msgpool_create_system() failed. (error code 0x%x)\n",
        _task_get_error() );

if ( _msgpool_create_system( sizeof( LINE_INFO_MSG ), 16L, 0L, 0L ) == FALSE )
    _io_printf("Peak_Data_Task: _msgpool_create_system() failed. (error code 0x%x)\n",
        _task_get_error() );

// allocate an 8MB struct! This is the incoming data buffer.
ref_peak_ptr = peak_data_ptr =
    (PEAK_DATA_BUFFER_PTR)_mem_alloc_system( sizeof(PEAK_DATA_BUFFER) );
if (peak_data_ptr == NULL)
{
    _io_printf("Peak_Data_Task: _mem_alloc_system() failed. (error code 0x%x)\n",
        _task_get_error() );
    puts("Fatal error, program is terminating.\n");
    _mqx_exit(0);
}

_io_printf("Data pointer: 0x%x\n", peak_data_ptr); // for debugging

// allocate info struct
info_ptr = (PEAK_DATA_INFO_STRUCT_PTR)
    _mem_alloc_system( sizeof( PEAK_DATA_INFO_STRUCT ) );
if (info_ptr == NULL)
{
    _io_printf("Peak_Data_Task: _mem_alloc_system() failed. (error code 0x%x)\n",
        _task_get_error() );
    puts("Fatal error, program is terminating.\n");
    _mqx_exit(0);
}

// initialize struct to default values
info_ptr->MODE = 0;                 // image/continuous
info_ptr->VOXEL_CNTR = 0;
info_ptr->VOXELS_PER_LINE = 0;
info_ptr->TOTAL_VOXELS = 0;
info_ptr->LINE_NUM = 0;
info_ptr->COLD_START = TRUE;
info_ptr->WARM_START = FALSE;
info_ptr->MOTION_DIRECTION = MOTION_DIR_BOTH;
```

peakdata.c continued

```
info_ptr->MOTION_DETECT_THRESHOLD = MOTION_DIFF_THRESHOLD;
info_ptr->MOTION_FILTER_CONSTANT = MOTION_K_FACTOR;
info_ptr->BUFFER_PTR = (uint_16 *)ref_peak_ptr;
info_ptr->REF_BUFFER_PTR = (uint_16 *)ref_peak_ptr;
// more... tbd

// for data transfers to file handler, (not used in 3DMD)
buffer_ptr = (uint_16 *)ref_peak_ptr;

// Initialize Interrupt Handling
// Assign FPort Rx_AF interrupt (0x2) to portable int. code 3
InterruptSource((int)EINT3_INTERRUPT, (int)2);

// install Peak_Data_Notifier()
vector = EINT3_INTERRUPT + 4; // offset by 4 from portable int code.
_int_install_isr( vector, (void (_CODE_PTR_)(pointer))Peak_Data_Notifier, (pointer)info_ptr);

// use no interrupt polarity inversion for fifoport operation
INTR_EXT_POLARITY(3,0); // EXT_INT7
ClearInterrupt((int)EINT3_INTERRUPT);
EnableInterrupt((int)EINT3_INTERRUPT);

// create Peak notifier system queue.
peak_notifier_sys_qid = _msgq_open_system( PEAK_NOTIFIER_SYS_Q, 0, NULL, 0 );
if ( peak_notifier_sys_qid == 0 )
    _io_printf("Peak_Data_Task: Call to _msgq_open_system() failed. (code 0x%x)\n",
        _task_get_error() );

// create Motion_Detect_Task()
Motion_detect_task_id = _task_create( On_This_Processor, MOTION_DETECT_TASK, 0 );
if (Motion_detect_task_id == MQX_NULL_TASK_ID)
    _io_printf("Peak_Data_Task: Call to _task_create() failed. (code 0x%x)\n", _task_get_error() );

// main loop
while ( TRUE )
{
    /* wait for next incoming message */
    peak_cmd_msg_ptr = _msgq_receive( MSGQ_ANY_QUEUE, 0 );
    if( peak_cmd_msg_ptr == NULL )
        _io_printf("Peak_Data_Task: Call to _msgq_receive() failed. (code 0x%x)\n",
            _task_get_error() );

    switch (peak_cmd_msg_ptr->CMD)
    {
        case START_PEAK:
        {
            /* prepare a new message buffer */
            ack_msg_ptr = (UTILITY_MESSAGE_PTR)
                _msg_alloc_system( (uint_16)sizeof(UTILITY_MESSAGE) );
            if( ack_msg_ptr == NULL )
                _io_printf("Peak_Data_Task: Call to _msg_alloc_system() failed. (Error code 0x%x).\n",
                    _task_get_error() );

            // message from User Interface Task
            // reinitialize peak detector info struct
            info_ptr->MODE = 0;
            info_ptr->VOXEL_CNTR = 0;
            voxels = info_ptr->VOXELS_PER_LINE = peak_cmd_msg_ptr->VOXELS_PER_LINE;
            info_ptr->TOTAL_VOXELS = 0;
            info_ptr->LINE_NUM = 0;
            info_ptr->WARM_START = TRUE;
            info_ptr->FIRST_PASS = TRUE;
            info_ptr->MOTION_DIRECTION = peak_cmd_msg_ptr->MOTION_DIRECTION;
```

peakdata.c continued

```
    info_ptr->MOTION_FILTER_CONSTANT = peak_cmd_msg_ptr->
        MOTION_FILTER_CONSTANT;
    info_ptr->MOTION_DETECT_THRESHOLD = peak_cmd_msg_ptr->
        MOTION_DETECT_THRESHOLD;
    info_ptr->BUFFER_PTR = (uint_16 *)ref_peak_ptr;
    info_ptr->REF_BUFFER_PTR = (uint_16 *)ref_peak_ptr;
    // Reset FIFOPORT to eliminate peak stalls.
    // this is executed BEFORE the galvo is started.
    // so synchronization is guaranteed.
    enable_fifo_port();
    reset_fifo_port();

    // setting AF threshold requires a handshake with remote processor
    fifo_port_spit(INIT_FPORT); // AF threshold is set by remote processor

    // peak detector uses a 1ms delay to read the fifoport
    // wait here for an ack before proceeding
    result = fifo_port_key();
    if (result != PEAK_READY) // check for handshake problem.
        _io_printf("Peak_Data_Task(): Bad fifoport ack from Peak Detector.\n");

    // reply to user interface task and cleanup...
    ack_msg_ptr->HEADER.SIZE = (_msg_size)sizeof( UTILITY_MESSAGE_PTR );
    ack_msg_ptr->HEADER.TARGET_QID = peak_cmd_msg_ptr->HEADER.SOURCE_QID;
    ack_msg_ptr->HEADER.SOURCE_QID = peak_cmd_msg_ptr->HEADER.TARGET_QID;
    if ( _msgq_send( (pointer)ack_msg_ptr ) == 0 )
        _io_printf("Peak_Data_Task: Call to _msgq_send() failed. (Error code 0x%x).\n",
            _task_get_error());
    _msg_free( (pointer)peak_cmd_msg_ptr );

    break;
} /* Endcase */

case RQST_LINE:
{
    // a request from File_Handler_Task for image data (not used in 3DMD)
    line = peak_cmd_msg_ptr->LINE_NUM;
    // this msg allocation will be freed by the file handler task.
    peak_data_rply_msg_ptr = (PEAK_DATA_RPLY_MSG_PTR)
        _msg_alloc_system( sizeof( PEAK_DATA_RPLY_MSG ) );
    if ( peak_data_rply_msg_ptr == NULL )
        _io_printf("Peak_Data_Task: Error in call to _msg_alloc_system(). Code 0x%x\n",
            _task_get_error() );

    peak_data_rply_msg_ptr->HEADER.SIZE =
        sizeof( MESSAGE_HEADER_STRUCT ) + (voxels * 4 * NUMBER_OF_PEAKS);
    peak_data_rply_msg_ptr->HEADER.SOURCE_QID = peak_data_qid;
    peak_data_rply_msg_ptr->HEADER.TARGET_QID = file_qid;

    first_point = line * voxels;
    last_point = first_point + voxels;
    for ( i = 0; i < voxels; i++ )
    {   // build line data for message to file handler task
        peak_data_rply_msg_ptr->VOXEL_BUFF[i].VOXEL[0].PEAK_POSN =
            *(buffer_ptr + 2*(i + (line * voxels)));
        peak_data_rply_msg_ptr->VOXEL_BUFF[i].VOXEL[0].PEAK_INT =
            *(buffer_ptr + 2*(i + (line * voxels)) + 1);
    }

    if ( _msgq_send( peak_data_rply_msg_ptr ) == FALSE )
        _io_printf("Peak_Data_Task. Error in call to _msgq_send(). <Code 0x%x>\n",
            _task_get_error() );
```

peakdata.c continued

```
            // free rqst message.
                _msg_free( (pointer)peak_cmd_msg_ptr );

                break;
            } /* Endcase */

        } /* Endswitch */
    }

} /* Endbody */
```

motiondetect.c

```
/*****************************************************************
 *
 *   DESCRIPTION:  Motion Detection Task
 *
 *                 This task is created by the User_Interface_Task.
 *                 It requests messages from the Peak_Data_Notifier
 *                 containing a pointer to the latest line data.
 *                 It accesses this data and computes a running average
 *                 of range data for each point. It compares the new
 *                 data with the current average and flags any points
 *                 which exceed threshold (i.e. are in motion). Points
 *                 in motion are then sent to the Display_Driver_Task
 *                 for transfer to VB for display.
 *
 *                 All stdout debug/warning messages appear on a CCS output window.
 *
 *
 *   AUTHOR:       D.Green
 *
 *   HISTORY:      First Version 2001-08-28
 *                 Date          Who       What
 *                 2001-08-30    DG      added range averaging and difference detection
 *                 2001-11-22    DG      fixed bug in allocation of display_driver_msg
 *
 *
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 ******************************************************************/


/*TASK*-----------------------------------------------------------------
*
* Task Name : Motion_Detect_Task
* Comments  :
*
*END*-----------------------------------------------------------------*/


void Motion_Detect_Task
   (
      uint_32 parameter
   )

{ /* Body */

   int_32                        line_num;
   float                         Iavg[MAX_VOXELS_PER_LINE];
   int_16                        Idiff[MAX_VOXELS_PER_LINE];
   float                         k;

   int_16                        threshold; // must convert to Q6 format
   uint_32                       i;
   uint_16                       j;
   uint_32                       motion_type; // pos/neg/both
   int_16                        *buffer_ptr;
   uint_32                       voxels;

   _queue_id                     motion_detect_1_qid;
   _queue_id                     motion_detect_2_qid;
   _queue_id                     peak_notifier_sys_qid;
   _queue_id                     display_driver_qid;

   _pool_id                      display_driver_msg_pool_id;
```

motiondetect.c continued

```
_task_id                        display_driver_task_id;
UTILITY_MESSAGE_PTR             motion_detect_rqst_msg_ptr;
LINE_INFO_MSG_PTR               line_info_msg_ptr;
DISPLAY_DRIVER_MSG_PTR          display_driver_msg_ptr;
UTILITY_MESSAGE_PTR             display_driver_rqst_msg_ptr;
PEAK_DATA_INFO_STRUCT_PTR       info_ptr;


// initialize
for (i = 0; i < MAX_VOXELS_PER_LINE; ++i)
{
    Iavg[i] = 0;
    Idiff[i] = 0;
}

// banner (debug info)
_io_printf("Iavg = 0x%x \nIdiff = 0x%x\n", Iavg, Idiff);

peak_notifier_sys_qid = _msgq_get_id( On_This_Processor, PEAK_NOTIFIER_SYS_Q);

// create message queue for incoming notifier messages
motion_detect_1_qid = _msgq_open( MOTION_DETECT_Q, 0 );
if (motion_detect_1_qid == 0)
    _io_printf("Motion_Detect_Task(): Failure in call to _msgq_open() (code 0x%x)\n",
        _task_get_error() );

// create message queue for incoming display driver messages
motion_detect_2_qid = _msgq_open( MOTION_DETECT_2_Q, 0 );
if (motion_detect_2_qid == 0)
    _io_printf("Motion_Detect_Task(): Failure in call to _msgq_open() (code 0x%x)\n",
        _task_get_error() );

//create Display_Driver_Task
display_driver_task_id = _task_create( Q67_1A_ID, DISPLAY_DRIVER_TASK, 0 );
if (display_driver_task_id == MQX_NULL_TASK_ID)
    _io_printf("Motion_Detect_Task(): Failure in call to _task_create(), (code 0x%x)\n",
        _task_get_error() );

// create display driver message pool
display_driver_msg_pool_id = _msgpool_create( sizeof(DISPLAY_DRIVER_MSG), 16L, 16L, 0L );
if (display_driver_msg_pool_id == 0)
    _io_printf("Motion_Detect_Task: Failure in call to _msgpool_create() (code 0x%x)\n",
        _task_get_error() );

display_driver_qid = _msgq_get_id( Q67_1A_ID, DISPLAY_DRIVER_Q );

// prefetch display driver message from message pool
display_driver_msg_ptr = _msg_alloc( display_driver_msg_pool_id );
if (display_driver_msg_ptr == NULL )
    _io_printf("Motion_Detect_Task(): Failure in call to _msg_alloc() (code 0x%x)\n",
        _task_get_error() );

// main loop
while(TRUE)
{
    //prepare to send message to Peak_Data_Notifier to report to work
    motion_detect_rqst_msg_ptr = _msg_alloc_system( sizeof(UTILITY_MESSAGE) );
    if (motion_detect_rqst_msg_ptr == 0)
        _io_printf("Motion_Detect_Task(): Failure in call to _msg_alloc_system() (code 0x%x)\n",
            _task_get_error() );

    motion_detect_rqst_msg_ptr->HEADER.SIZE = sizeof( UTILITY_MESSAGE );
    motion_detect_rqst_msg_ptr->HEADER.SOURCE_QID = motion_detect_1_qid;
```

motiondetect.c continued

```
motion_detect_rqst_msg_ptr->HEADER.TARGET_QID = peak_notifier_sys_qid;

if (_msgq_send((pointer)motion_detect_rqst_msg_ptr) == FALSE)
    _io_printf("Motion_Detect_Task(): Failure in call to _msgq_send() (code 0x%x)\n",
        _task_get_error() );

// wait for the reply from Peak_Data_Notifier().
line_info_msg_ptr = _msgq_receive( motion_detect_1_qid, 0 );
if (line_info_msg_ptr == NULL)
    _io_printf("Motion_Detect_Task(): Failure in call to _msgq_receive() (code 0x%x)\n",
        _task_get_error() );

info_ptr = line_info_msg_ptr->INFO_STRUCT_PTR;
_msg_free( (pointer)line_info_msg_ptr );

// calculate position of data pointer and fetch line of data
voxels = info_ptr->VOXELS_PER_LINE;
line_num = info_ptr->LINE_NUM - 1;
if (line_num < 0)
    line_num = (MAX_NUM_OF_VOXELS / voxels) -1;  // wrap to largest line number

k = info_ptr->MOTION_FILTER_CONSTANT;
threshold = info_ptr->MOTION_DETECT_THRESHOLD * 64; // now converted to Q6
motion_type = info_ptr->MOTION_DIRECTION;

// pointer to current line of data (each voxel has 4 bytes)
buffer_ptr =
    (int_16 *)info_ptr->REF_BUFFER_PTR + (2 * line_num * voxels * NUMBER_OF_PEAKS);

j = 0;
display_driver_msg_ptr->DATA.COUNT = 0; // initialize

// on the first pass, initialize Iavg[] and clear Idiff[] to prevent spurious motion artifacts.
if (info_ptr->FIRST_PASS)
{
    info_ptr->FIRST_PASS = FALSE;
    for (i = 0; i < voxels; ++i )
    {
        Iavg[i] = *(buffer_ptr + 2*i);
        Idiff[i] = 0;
    } //Endfor
}

else
{
    // process the latest line of data
    for (i = 0; i < voxels; ++i )
    {
        // compute running average for each image point
        // note that range and intensity samples are interleaved
        // compute range departure from average for each point
        // Idiff is in Q6 format
        if (*(buffer_ptr + 2*i) > 0)
        {
            Iavg[i] = ((1. - k) * Iavg[i]) + (k * (float)*(buffer_ptr + 2*i));
            Idiff[i] = (int_16)((float)*(buffer_ptr + 2*i) - Iavg[i]);
        }
        else
            Idiff[i] = 0;
        //Endif
```

motiondetect.c continued

```
            if (motion_type == MOTION_DIR_POS)
            {
                if ( ((threshold > 0) && ((Idiff[i] - threshold) > 0))
                    || ((threshold < 0) &&   (Idiff[i] + threshold ) > 0 ))
                {
                    display_driver_msg_ptr->DATA.INDEX[j] = i;
                    display_driver_msg_ptr->DATA.VALUE[j] = Idiff[i];
                    display_driver_msg_ptr->DATA.COUNT = ++j;
                } //Endif
            }

            else if (motion_type == MOTION_DIR_NEG)
            {
                if ( ((threshold < 0) && ((Idiff[i] - threshold) <  0))
                    || ((threshold > 0) && (Idiff[i] + threshold) < 0))
                {
                    display_driver_msg_ptr->DATA.INDEX[j] = i;
                    display_driver_msg_ptr->DATA.VALUE[j] = Idiff[i];
                    display_driver_msg_ptr->DATA.COUNT = ++j;
                } //Endif
            }

            else    // motion_type is 'BOTH'
            {
                if ( ((threshold > 0) && ((abs(Idiff[i]) - threshold) > 0))
                    || ((threshold < 0) && ((-abs(Idiff[i]) - threshold) < 0)))
                {
                    display_driver_msg_ptr->DATA.INDEX[j] = i;
                    display_driver_msg_ptr->DATA.VALUE[j] = Idiff[i];
                    display_driver_msg_ptr->DATA.COUNT = ++j;
                } //Endif
            } //Endif
        } //Endfor
    } //Endif

    // check for request from display driver
    display_driver_rqst_msg_ptr = _msgq_poll( motion_detect_2_qid );
    if (display_driver_rqst_msg_ptr != NULL)
    {
        if (display_driver_rqst_msg_ptr->DATA == RQST_NEXT_DATA)
        {
            // for this line, send indices and departures to Display_Driver_Task()
            display_driver_msg_ptr->HEADER.SIZE = sizeof( DISPLAY_DRIVER_MSG );
            display_driver_msg_ptr->HEADER.SOURCE_QID = motion_detect_2_qid;
            display_driver_msg_ptr->HEADER.TARGET_QID = display_driver_qid;
            display_driver_msg_ptr->TYPE = MOTION;
            if ( _msgq_send( display_driver_msg_ptr ) == FALSE )
                _io_printf("Motion_Detect_Task(): Failure in msgq_send() (code 0x%x)\n",
                    _task_get_error() );

            // dump the message and continue
            _msg_free( (pointer)display_driver_rqst_msg_ptr );

            // prefetch next display driver message from message pool
            display_driver_msg_ptr = _msg_alloc( display_driver_msg_pool_id );
            if (display_driver_msg_ptr == NULL )
                _io_printf("Motion_Detect_Task(): Failure in call to _msg_alloc() (code 0x%x)\n",
                    _task_get_error() );

        } //Endif
    } //Endif

  } //Endwhile
} /* Endbody */
```

iiq67.cmd

```
// Linker command file for Q67.

/*  This version explicitly allocates SDRAM for large data buffers
    which are required for the laser scanner.
*/

/* SPECIFY THE SYSTEM MEMORY MAP */

/* MAP 0 */
/*
MEMORY
{
   SBSRAM:      o = 00000000h  l = 00040000h  Kernel data
   IPM:         o = 01400000h  l = 00010000h
   INT:         o = 02000000h  l = 00000200h  Interrupt vectors
   SDRAM0:      o = 02000200h  l = 003FFE00h
   SDRAM1:      o = 03000000h  l = 00400000h
   IDM:         o = 80000000h  l = 00010000h
}
*/

/* MAP 1 - modified for IIQ67 */
MEMORY
{
   IPM:         o = 00000000h  l = 00010000h   /* program cache */
   IDM:         o = 80000000h  l = 00010000h   /* internal data memory */
   INT:         o = 02000000h  l = 00000200h   /* Interrupt vectors */
//  SDRAM0:     o = 02000200h  l = 00fffdffh
    SDRAM0A:    o = 02000200h  l = 006ffdffh
    SDRAM0B:    o = 02700000h  l = 008fffffh   /* for large data buffers */
    ASRAM       o = 01700000h  l = 00080000h   /* 128k x 32 (512kB) For PCI interface */
}

/*
With the exception of .text, the initialized and uninitialized sections cannot
be allocated into internal program memory.
Initialized sections:
.cinit, .const, .switch section, and .text
Uninitialized sections:
.bss section, .far section, .stack section, .sysmem
*/

SECTIONS
/*
{     for testing, put all sections in onchip memory
   .text      >     IPM
   .vectors   >     IPM     Must be in same section as .text
   .stack     >     IDM
   .bss       >     IDM
   .cinit     >     IDM
   .cio       >     IDM
   .const     >     IDM
   .data      >     IDM
   .switch    >     IDM
   .sysmem    >     IDM
   .far       >     IDM
   .ipmtext   >     IDM
   .vec       >     IDM
}
*/
```

iiq67.cmd continued

```
{   /* IPM is reserved for program cache */
   .text        >       SDRAM0A
   .vectors     >       INT         /* Must be in same section as .text */
   .stack       >       IDM
   .bss         >       IDM
   .cinit       >       SDRAM0A
   .cio         >       IDM
   .const       >       IDM
   .data        >       IDM
   .switch      >       IDM
    .sysmem     >       SDRAM0B
   .far         >       SDRAM0A
   .ipmtext     >       SDRAM0A
   .vec         >       SDRAM0A
}
```

demo0A.mki

```
# make "include file"

#
# Generic tools file for *.MAK
#

#
# Macros
#
ASM = asm6x
ASM_ARGS = -s -e
CC = cl6x
CC_ARGS = -g -q -x2 -o2 -me -ml2
LNK = lnk6x
LNK_ARGS = -c -x -stack 0x800 -heap 0x0800000
AR = ar6x
AR_ARGS = -r
HEX = hex6x
HEX_ARGS = prom.cmd
HEX2BIN = hex2bin
HEX2BIN_ARGS = -f
HEXVERT = hexvert
HEXVERT_ARGS = -w4 -r
LIBS = -l stdio.lib -l periph.lib -l user.lib -l dsp.lib -l rts6701le.lib -l iiq67.lib -l mqx.lib
DEBUGGER = c:\composer\cc_app.exe
# EXECUTE = terminal.exe

# Uncomment to explicitly set output object file
# OUTPUT = filename.out
```

Appendix B. Code listings for the Peak M62 and Galvo M62 DSPs

This appendix lists the 3DMD code for the Peak M62 and Galvo M62 DSPs. This code runs standalone, without the Precise/MQX real-time operating system.

For each DSP, the listings include the “selector” file, the source files, the linker command file and the “make include file”.

Code62A.c

```
/* Peak M62 selector file                    */
#include "c:\m6x\include\target\periph.h"
#include "c:\m6x\include\target\stdio.h"
#include "c:\m6x\include\target\user.h"

/* application includes */
#include "c:\Working\WaveformDemos\Demo5\peak5\iim62\template.h"
#include "c:\Working\WaveformDemos\Demo5\peak5\iim62\pdfn.h"
#include "c:\Working\WaveformDemos\Demo5\peak5\iim62\peak.h"
#include "c:\Working\WaveformDemos\Demo5\peak5\iim62\demo.h"
#include "c:\Working\WaveformDemos\Demo5\peak5\iim62\externs.c"
#include "c:\Working\WaveformDemos\Demo5\peak5\iim62\main.c"
#include "c:\Working\WaveformDemos\Demo5\peak5\iim62\peak_isr.c"

/*-----------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------------*/
```

templates.h

```
// Prototypes
#pragma INTERRUPT(Peak_isr);
void Peak_isr(void);

/*-----------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------*/
```

pdfn.h

```
# ifndef _PDFN_
# define _PDFN_

/*H*********************************************************************
 *  File     : pdfn.h          ;
 *  Date     : 99-11-12          ;
 *  Rev      : 1.0             ;
 *  By       : F.Blais, D.Green     ;
 *
 *  Description :
 *      Multiple PeakDetector Definitions
 *
 *      The Multiple Peak Detection Algorithm has been used intensively
 *      in numerous Biris and AutoSynchronized Scanners. This file contains
 *      the definitions required to implement a software version of this
 *      algorithm. Provision are made available to add other detected
 *      parameters to this algorithm
 *
 *  NOTES:
 *      1) For speed reason, the C code uses extensively GLOBAL variables and
 *      constant definitions to access registers and data. These are defined
 *      as G_PKDT_INDATA_
 *      2) The peak detector algorithm can use two different methods to
 *      read the CCD signal, either from a buffer or from a register. This
 *      is defined using the G_PKDT_INDATA and G_PKDT_INIT_INPUT
 *
 *
 * Rev Date          By     Comments
 * ---------------------------------------------------------------
 * 1.0  99-11-12  fb      Release Version 1.0
 * 1.1  00-05-23  DG      typedef for pkdt_data_struct now uses ints
 * 1.2  00-05-26  DG      use of union for ccd_buffer[]
 *                        added ping-pong buffering for concurrent dma
 *
 **********************************************************************
 */
/*-------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-------------------------------------------------------------------*/


/*
    Global System Definitions:

    These definitions must be adapted to the system
*/

/* External variables */
//extern volatile int  *fifo_ptr;       /* From FIFO */
//extern volatile int  *ccd_buffer;

/**** GLOBAL MACROS *****/
/* Must be modified to reflect the system */

# define PKDT_DEBUG    0              /* 1 = Debug Mode Active */


/* Read the data from the A/D convecter */
/*
    Two modes are defined: buffered or register.
    IN BOTH CASES THE POINTER OR FIFO REGISTER MUST BE PROPERLY INITIALIZED
    PRIOR TO CALLING THE PEAK DETECTOR ALGORITHM.
```

pdfn.h continued

```
    Note:
    for C6x operation, dma is used to transfer data directly to
    onchip RAM. Therefore the buffer mode, as shown below, is used.
*/

#define G_PKDT_INDATA_TYPE      1       /* Set mode: 0=Fifo, 1=Buffered */

#define G_PKDT_INDATA_FIFO      *fifo_ptr & 0x3fff

// for M62
// double buffering is used because dma and peak detection operate concurrently.
// these macros correct ADC data
#define G_PKDT_INDATA_BUFFER_0(i)
    (((ccd_data_0[i].data_pr.ad0 & 0x3fff) + A4D1_ADC_OFFSET) * A4D1_ADC_SIGN)
#define G_PKDT_INDATA_BUFFER_1(i)
    (((ccd_data_1[i].data_pr.ad0 & 0x3fff) + A4D1_ADC_OFFSET) * A4D1_ADC_SIGN)


/***** LOCAL DEFINITIONS *****/
#define PKDT_CCD_MAX_LENGTH     128             /* Maximum length peak detector */
#define PKDT_PIX_QFORMAT        6
#define PKDT_PIX_FRACTION       (1 << PKDT_PIX_QFORMAT)

#define PKDT_PIX_POS(p)         ((p) / (float)PKDT_PIX_FRACTION)


/*  NOTE: The following structure uses ONLY 32 bits values ALL long integers. That
    prevents non-compatible formats between TI and IEEE floating point values.

    NB (2000-05-23, DG.):
    for the C6x, long is 40 bits not 32. so the following typedef should be
    changed to use ints for the reasons stated above.

    (2001-03-22, DG.):
    use 'short' declarations for efficiency.
*/


typedef struct {
    short   lPeakPosQ6;         /* Peak position (Format Q) */
    short   lPeakInt;
    short   lPeakSigma;
    short   lNotUsed;
} pkdt_data_struct;


/* Prototypes */

/*
    iInitPeakDetect
    iPeakDetect
        Detect ALL peaks in the CCD signal to a maximum of iMaxNbPeak. Stores
        the results in the tPkDt structure to a maximum of iMaxNbPeak.
        Skips iSkipData and then reads the remaining iCCDLength.

        The MACROS G_PKDT_INDATA and G_PKDT_INDATA_INIT must be properly defined

        The function iInitPeakDetect MUST BE CALLED AT LEAST ONCE before
        calling iPeakDetect.

        The MACRO G_PKDT_INDATA_INIT MUST BE CALLED BEFORE EACH call to iPeakDetect.

    Inputs
        iCCDLength      Number of pixel on the CCD (total values of data read)
```

pdfn.h continued

```
        iCCDWindowFromStarts detecting signal at specified location
        iCCDWindowLength   Detect for ... skip remaining pixels
        iValidThreshold        Validation threshold
        ptPkDt          Pointer where to store data
        iMaxNbPeak        Maximum number of peaks in the buffer
*/

int iInitPeakDetect (int iCCDLength, int iCCDWindowFrom, int iCCDWindowLength,
    short iValidThreshold);
int iInitPeakDetectReg (int iCCDLength, int iCCDWindowFrom, int iCCDWindowLength,
    int iValidThreshold);

int iPeakDetect (pkdt_data_struct *ptPkDt, int iMaxNbPeak);
int iPeakDetectReg (pkdt_data_struct *ptPkDt, int iMaxNbPeak);



/*
    In Debug mode only
*/

# if PKDT_DEBUG
extern int aiCCD_filter_signal [PKDT_CCD_MAX_LENGTH];
extern int aiCCD_derive_1st [PKDT_CCD_MAX_LENGTH];
extern int aiCCD_derive_2nd [PKDT_CCD_MAX_LENGTH];
extern int aiDebug [PKDT_CCD_MAX_LENGTH];
# endif


# endif /* _PDFN_ */
```

peak.h

```
/*
 *  Rev    Date          By        Comments
 * ---------------------------------------------------------------------
 *  1.0    2001-01-31    d.g.      Initial release
 * ---------------------------------------------------------------------
 *
 *  Description:
 *              Header file for Peak ISR.
 *
 */

/*---------------------------------------------------------------------
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *---------------------------------------------------------------------*/



#ifndef    __Peak_h__
#define    __Peak_h__

// DIO bits
#define DIO_PIN_1       0x0
#define DIO_PIN_2       0x1
#define DIO_PIN_3       0x2
#define DIO_PIN_4       0x3
#define DIO_PIN_5       0x4
#define DIO_PIN_6       0x5
#define DIO_PIN_7       0x6

#define    IDLE         0
#define    RUN          1

#define MODULE_0        0
#define MODULE_1        1

#define A4D1_SITE       0L
#define A4D4_SITE       0L

#define ADC_PAIR_0      0
#define ADC_PAIR_1      1

#define ADC_FIFOs       1
#define DAC_FIFOs       2

// ADC pipeline delay
#define ADC_LATENCY          3

#define CCD_LENGTH           128
#define MAX_NB_PEAK          6

// DMA: cpu block/no-block
#define BLOCK           1
#define NO_BLOCK        0

#endif /* Peak_h */
```

demo.h

```
#ifndef  _demo_h_
#define  _demo_h_

#define full_version 0    // #ifdef used for early development

/*
    this file is copied from the Q67. It is used only to provide consistency between
    the Q67 and M62 systems.
*/

/* Global indicies: task template numbers. */
#define IPC_TTN                    10
#define USER_INTERFACE_TASK        11
#define GALVO_CONTROL_TASK         15
#define PEAK_DATA_TASK             20
#define FILE_HANDLER_TASK          25
#define GALVO_DATA_TASK            30

/* Queues - <#1 - 7 are reserved> */
#define USER_INTERFACE_Q       40
#define GALVO_CONTROL_Q        41
#define GALVO_DATA_Q           42
#define PEAK_DATA_Q            43
#define FILE_HANDLER_Q         44

/* Interprocessor message queues */
/* Q67 FIFOLinks - Board 1 */
// ProcA
#define    FLINK1_A_B_Q       10
#define    FLINK1_A_C_Q       11
#define    FLINK1_A_D_Q       12
// ProcB
#define    FLINK1_B_A_Q       13
#define    FLINK1_B_C_Q       14
#define    FLINK1_B_D_Q       15
// ProcC
#define    FLINK1_C_A_Q       16
#define    FLINK1_C_B_Q       17
#define    FLINK1_C_D_Q       18
// ProcD
#define    FLINK1_D_A_Q       19
#define    FLINK1_D_B_Q       20
#define    FLINK1_D_C_Q       21

/* Q67 FIFOLinks - Board 2 (future expansion) */
// ProcA
#define    FLINK2_A_B_Q       22
#define    FLINK2_A_C_Q       23
#define    FLINK2_A_D_Q       24
// ProcB
#define    FLINK2_B_A_Q       25
#define    FLINK2_B_C_Q       26
#define    FLINK2_B_D_Q       27
// ProcC
#define    FLINK2_C_A_Q       28
#define    FLINK2_C_B_Q       29
#define    FLINK2_C_D_Q       30
// ProcD
#define    FLINK2_D_A_Q       31
#define    FLINK2_D_B_Q       32
#define    FLINK2_D_C_Q       33

/* Processor numbers */
/* Q67 Board 1 */
```

demo.h continued

```
#define Q67_1A_ID          1
#define Q67_1B_ID          2
#define Q67_1C_ID          3
#define Q67_1D_ID          4

/* Q67 Board 2 */
#define Q67_2A_ID          5
#define Q67_2B_ID          6
#define Q67_2C_ID          7
#define Q67_2D_ID          8

/* M62 Boards (NOT REQUIRED) */
#define M62_1_ID           101    // galvo processor
#define M62_2_ID           102    // future expansion

/* misc */
#define On_This_Processor          0
#define MAX_BUFFERS                16

/* DMA related */
#define DMA_CH_3                   3 // DMA channel for FIFOPort access
#define BLOCK                      1
#define NO_BLOCK                   0

/* Galvo command type */
#define STOP_GALVO                 0x1010
#define START_GALVO                0x2020
#define UPDATE_GALVO               0x3030

/* Galvo misc */
#define    GALVO_ACK               0x4040
#define GALVO_OK                   0x1111
#define BAD_GALVO_RESPONSE         0x2222
#define NO_GALVO_RESPONSE          0x3333
#define GALVO_NOT_READY            0x4444

/* Galvo run mode */
#define CONTINUOUS_MODE            0
#define IMAGE_MODE                 0x1

/* Peak commands */
#define START_PEAK                 0x5
#define INIT_FPORT                 0x5555

/* Peak acknowledment */
#define PEAK_READY                 0x1010

/* No. of Peaks to Detect */
#define    NUMBER_OF_PEAKS         0x1

/* File handler message types */
#define    OPEN_FILE               0x0
#define WRITE_DATA                 0x1
#define CLOSE_ALL_FILES            0x2

/* File handler data types */
#define    POSITION                0x0
#define INTENSITY                  0x1

/* file status */
#define FILES_OPEN                 0x1
#define FILES_CLOSED               0x0

/* Utility message Data types */
```

demo.h continued

```
#define     NO_WRITE            0x1
#define  OPEN_FAIL              0x2

#endif
/*-----------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------------*/
```

externs.c

```
/*
 *  Rev     Date            By          Comments
 * ---------------------------------------------------------------------------
 *  1.0     2001-01-31   d.g.        Initial release
 * ---------------------------------------------------------------------------
 *
 *  Description:
 *
 *      external declarations
 *
 */

/*---------------------------------------------------------------------------
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *---------------------------------------------------------------------------*/


// externals for peak detector
typedef struct
{
    unsigned short      ad1;
    unsigned short      ad0;
} ADC_PAIR;

typedef union
{
    unsigned int        pair;   // dma transfer, 2 channels
    ADC_PAIR            data_pr;// data read, 1 channel
} CCD_DATA;

// ping-pong buffers
CCD_DATA        ccd_data_0[CCD_LENGTH];
CCD_DATA        ccd_data_1[CCD_LENGTH];

static int          in_buffer_select; // int for efficiency
static int          first_pass;


// used 'short' for performance testing
short               aiCCD_derive_2nd [PKDT_CCD_MAX_LENGTH];
```

main.c

```
/*
 *  Rev    Date          By        Comments
 * ---------------------------------------------------------------------
 *  1.0    2000-05-30    d.g.      Initial release
 *         2001-03-22    d.g.      adjusted for 'short' data
 *         2001-08-30    DG        added ack at end of INIT_FPORT
 * ---------------------------------------------------------------------
 *
 *  Description:
 *
 *      This test code does not run under MQX.
 *      Peakdetector setup program.
 *
 *
 *  Parameters:
 *
 *  Returns:
 *
 */

/*----------------------------------------------------------------------------
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2000, 2001
 *-----------------------------------------------------------------------------*/


#include "c:\m6x\include\target\periph.h"
#include "c:\m6x\include\target\stdio.h"
#include "pdfn.h"

void main()

{
    short                       ccd_threshold = 300;
    int                         module = A4D1_SITE;
    extern CCD_DATA             ccd_data_0[CCD_LENGTH];
    extern CCD_DATA             ccd_data_1[CCD_LENGTH];

    clrscr();

    //  Setup digital I/O direction MSBy...LSBy
    dig_dir(DIO_DIR_INPUT, DIO_DIR_INPUT, DIO_DIR_INPUT, DIO_DIR_OUTPUT);

    // clear DIG output FIFO ISR marker bit (pin 6)
    write_dig_bit(DIO_PIN_6, 0);

    enable_fifo_port(); // initialize fport
    reset_fifo_port();

    puts("Peak5\n-----\n");
    puts("\nPeak Detector. \n");
    puts("**** Fully optimized version ****\n");
    puts("DIG I/O pin 6 is asserted for the duration of the Peak_isr.");

/*
    This program uses the external CCD Clock to clock the A4D1.
*/

    // setup A4D1 Omnibus ADC module
    A4D1_initialize(module);
//  A4D1_gate(module, OFF);   // use hardware control
    A4D1_reset_fifo(module, 0); // adc fifos
    A4D1_trigger_adc(module, EXTERNAL); // use CCD Clock
    A4D1_set_fifo_interrupt_level(module, 0, A4D1_FIFO_HALF_FULL);
```

main.c contnued

```
    // enable fifo (Software control)
//   A4D1_set_gate_polarity(module, A4D1_RISING_EDGE, A4D1_FALLING_EDGE); // Hardware control
    A4D1_enable_fifo(module, 0, ON);          // Start filling ADC FIFO 0
    A4D1_enable_fifo(module, 1, OFF);         // Disable filling ADC FIFO 1
    A4D1_enable_fifo(module, 2, OFF);         // Disable filling DAC FIFO
    A4D1_gate(module, ON);                    // Use software control

    // setup interrupt
    EnableInterrupts();
    InterruptSource(EINT0_INTERRUPT, EINT0_INT_SRC);
    InstallIntVector(Peak_isr, EINT0_INTERRUPT);
    ClearInterrupt(EINT0_INTERRUPT);
    EnableInterrupt(EINT0_INTERRUPT);

    // initialize Peak detector
    iInitPeakDetect (CCD_LENGTH, 16, CCD_LENGTH - 2 * 16, ccd_threshold);

    printf("ccd_data_0: 0x%x\n", ccd_data_0);
    printf("ccd_data_1: 0x%x\n", ccd_data_1);

    enable_clock();
    while (1)
    {
    /*
    This loop checks for an INIT_FPORT command from the Peak_Data_Task().
    */
        ms(1); // awake every millisecond
        if ( (get_fifo_port_status() & Rx_FIFO_EMPTY) == 1 ) // if (not empty)...
        {
            switch (fifo_port_eat())
            {
                case INIT_FPORT:
                {   // set Q67 FPort threshold to Half full condition.
                    // this requires a handshake between the Q67 and the M62
                    set_fifo_port_AF_levels( 0, 252 );
                    ClearInterrupt(EINT0_INTERRUPT);

                    // setup A4D1
                    A4D1_initialize(module);
                    A4D1_reset_fifo(module, 0); // adc fifos
                    A4D1_trigger_adc(module, EXTERNAL); // use CCD Clock
                    A4D1_set_fifo_interrupt_level(module, 0, A4D1_FIFO_HALF_FULL);

                    // enable fifo (Software control)
                   A4D1_enable_fifo(module, 0, ON);       // Start filling ADC FIFO 0
                   A4D1_enable_fifo(module, 1, OFF);      // Disable filling ADC FIFO 1
                   A4D1_enable_fifo(module, 2, OFF);      // Disable filling DAC FIFO
                    A4D1_gate(module, ON);                    // Use software control

                    // [2001-08-30] send ack to Q67
                    fifo_port_put16(PEAK_READY);
                    break;
                } /* Endcase */

// tbd        case :
//              break;

            } /* Endswitch */
        } /* Endif */
    } /* Endwhile */
}
```

peak_isr.c

```
/*
 *  Rev    Date          By        Comments
 * ---------------------------------------------------------------------
 *  1.0    2001-02-02    d.g.      Initial release
 * ---------------------------------------------------------------------
 *
 *  Description:
 *
 *      This test code does not run under MQX.
 *      Peakdetector interrupt service routine.
 *
 *
 *  Parameters:
 *
 *  Returns:
 *
 */

/*---------------------------------------------------------------------
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *---------------------------------------------------------------------*/


#include "c:\m6x\include\target\periph.h"
#include "c:\m6x\include\target\stdio.h"
#include "pdfn.h"


/*
 *  FUNCTION: Peak_isr
 *
 *  PARAMETERS:
 *
 *  DESCRIPTION:
 *              Interrupt Service Routine triggered by A4D1's half-full (HF) interrupt.
 *
 *  RETURNS:
 *              none
 */

void Peak_isr (void)
{ //Body

/* declarations required by external peak detector function */
//  extern short            aiCCD_derive_2nd[PKDT_CCD_MAX_LENGTH];
    extern CCD_DATA         ccd_data_0[CCD_LENGTH];
    extern CCD_DATA         ccd_data_1[CCD_LENGTH];

    extern int              in_buffer_select;

    pkdt_data_struct        atPkDt[MAX_NB_PEAK];
    int                     nbPeak;
    A4D1                    *a4d1 = (A4D1*)&Periph->Module[0];

    int                     i;
    const int               multi_peak_num = NUMBER_OF_PEAKS;

//  pkdt_data_struct volatilePeak_data[MAX_NB_PEAK];

    // set marker pin 6 – for performance measurement
    write_dig_bit( DIO_PIN_6, 1 );
```

peak_isr.c continued

```
    if (in_buffer_select == 0)    // choose buffer
        // fetch data from A4D1
        dma_port_to_mem(0, (int*)&a4d1->adc[0].both, (int*)ccd_data_0, CCD_LENGTH, NO_BLOCK);

    else
        dma_port_to_mem(0, (int*)&a4d1->adc[0].both, (int*)ccd_data_1, CCD_LENGTH, NO_BLOCK);


    // toggle buffer selector before call to iPeakDetect()
    if (in_buffer_select == 0)
        in_buffer_select = 1;

    else
        in_buffer_select = 0;

/* non-registered version */
    nbPeak = iPeakDetect (atPkDt, multi_peak_num);

    if ( nbPeak == 0 )
        nbPeak = 1;  /* to satisfy the for loop below */

    /* for testing: only store data for single point */
    for ( i = 0; i < nbPeak; i++ )
    {   // send to FIFOPort, no waiting
        // data converted to 'short' (2001-03-22)
        fifo_port_spit( atPkDt[i].lPeakPosQ6 );
        fifo_port_spit( atPkDt[i].lPeakInt );
    } //Endfor

   ClearInterrupt(EINT0_INTERRUPT);

    // clear marker pin 6
    write_dig_bit( DIO_PIN_6, 0 );
} //Endbody


/*C*********************************************************************
 *  File     : PeakDet.c            ;
 *  Date     : 99-11-12             ;
 *  Rev      : 1.0              ;
 *  By       : F.Blais, D.Green     ;
 *
 *  Description :
 *      Multiple PeakDetector Algorithm
 *
 *      This file contains the Multiple Peak Detection Algorithm software
 *      implementation. The code has been optimized for speed and tries to
 *      minimize the memory requirements by using as little as possible of
 *      data memory which might be very important for DSP implementation.
 *
 *      Provisions are made to add other detected parameters to this
 *      algorithm such as the Sigma variable
 *
 *
 *  NOTES:
 *      1) For speed reasons, the C code uses GLOBAL variables and constant
 *      definitions to access registers and data. These are defined
 *      as G_PKDT_INDATA_ in peakdet.h file.
 *      2) The peak detector algorithm can use two different methods to
 *      read the CCD signal, either from a buffer or from a register. This
 *      is defined using the G_PKDT_INDATA and G_PKDT_INIT_INPUT
 *
 *  This version of the code uses GLOBAL arrays to store a copy of the internal
 *  processing operations IN THE DEBUG MODE ONLY. This allows the use
```

peak_isr.c continued

```
 *  internal/local variables for the "run-time" implementation and therefore
 *  to optimise speed.
 *
 * Rev Date          By     Comments
 * ----------------------------------------------------------------
 * 2.0  99-11-19     fb     Release Version 2.0
 *                          Modified to illustrate the "real" processing delay
 *                          (Z transform)
 * 2.1  99-12-02     DG     minor changes for actual DSP implementation
 *                          see lines: 44, 83-89
 *
 * 2.2  2000-05-23   DG     changes for C6x implementation:
 *                          - DATA_SECTION
 *
 * 2.3  2000-05-25   DG     changes for C6x code optimization:
 *                          - use of 'int' rather than 'long' for pkdt_data_struct
 *                          - use of 'short' rather than 'int' for CCD and
 *                          derivative data.
 *                          - use of 'const' qualifier
 *
 * 2.4  2000-05-29   DG     - changes to support double buffering as required for
 *                          concurrent dma and peak detector operation.
 *                          - added compensation for ADC pipeline latency
 *
 * 2.5  2001-03-16   DG     - reverted to 'int' from 'short' for CCD and derivative data
 *                          - removed compensation for ADC pipeline latency.
 *
 * 2.6  2001-03-21   DG     - use of 'short' declaration for computation and transfers
 *
 ************************************************************************
 */


#include "pdfn.h"

/***** Peak Detector Filter Types *****/
/*
    Different filters have been used to detect peaks and perform validation
    These filters order are:
        2 = [1 0 -1]              Extra short filter (
        4 = [1 1 0 -1 -1]         Short Filter
        7 = [1 2 2 1 -1 -2 -2 -1]Medium length filter
        8 = [1 1 1 1 0 -1 -1 -1 -1]  Long filter
*/


#define FIR_D1_ORDER        8       /* 1st Derivative FIR filter (4 or 8) */
#define FIR_D2_ORDER        4       /* 2nd Derivative FIR Filter (2 or 4) */

#define FIR_MAX_ORDER       8

#define FIR_INT_LENGTH      ((FIR_D1_ORDER > 4) ? 4 : 2)
#define FIR_D1_TAP          (FIR_D1_ORDER/2 + 1)
#define FIR_D2_TAP          (FIR_D2_ORDER/2 + 1)


/* Macro for speed */

#if (FIR_D1_ORDER == 4)     /* [1 1] */
#define FIL_INTENSITY(buf)  (buf[0] + buf[1])
#elif (FIR_D1_ORDER == 8)   /* [1 1 1 1] */
#define FIL_INTENSITY(buf)  (buf[0] + buf[1] + buf[2] + buf[3])
#endif
```

peak_isr.c continued

```
# if (FIR_D2_ORDER == 4)        /* [1 1 0 -1 -1] */
# define FIL_DERIVE(buf)        ((buf[i] + buf[i-1]) - (buf[i-3] + buf[i-4]))
# elif (FIR_D2_ORDER == 2)      /* [1 0 -1] */
# define FIL_DERIVE(buf)        (buf[i] - buf[i-2])
# endif

/* Force the following variables into the internal memory for C4x (manual 3-7) */
// ifdef changed to "C4X"
# ifdef _C4X
# pragma DATA_SECTION(aiSignal,".onchip")
# pragma DATA_SECTION(aiFilterSignal,".onchip")
# pragma DATA_SECTION(aiDerive1st,".onchip")
# pragma DATA_SECTION(iThreshold,".onchip")
# pragma DATA_SECTION(iSkipFrom,".onchip")
# pragma DATA_SECTION(iLength,".onchip")
# pragma DATA_SECTION(iSkipEnd,".onchip")
# endif

// use short for efficiency
static short    aiSignal [FIR_INT_LENGTH];
static short    aiFilterSignal [PKDT_CCD_MAX_LENGTH];
static short    aiDerive1st [PKDT_CCD_MAX_LENGTH];
static short    iThreshold;
static int      iSkipFrom, iLength, iSkipEnd;
/*
    iPeakDetect
        Detect ALL peaks in the CCD signal to a maximum of iMaxNbPeak. Stores
        the results in the ptPkDt structure to a maximum of iMaxNbPeak.
        Skips iSkipData and then reads the remaining iCCDLength.

      The MACROS G_PKDT_INDATA and G_PKDT_INDATA_INIT must be properly defined

        The function iInitPeakDetect MUST BE CALLED AT LEAST ONCE before
        calling iPeakDetect.

        The MACRO G_PKDT_INDATA_INIT MUST BE CALLED BEFORE EACH call to iPeakDetect.

    Inputs
        iCCDLength          Number of pixels on the CCD (total values of data read)
        iCCDWindowFromStarts detecting signal at specified location
        iCCDWindowLength    Detect for ... skip remaining pixels
        iValidThreshold     Validation threshold
        ptPkDt          Pointer where to store data
        iMaxNbPeak          Maximum number of peaks in the buffer
*/


/*
    iInitPeakDetect
        Initialize global variables (for speed) and check if parameters
        are "valid" for this CCD. Automatically adjust the "detection window"
        to also include the FIR filter processing length and delays

        This function can be called only once.

    Returns 1 if OK, 0 if error
*/


int iInitPeakDetect (int iCCDLength, int iCCDWindowFrom, int iCCDWindowLength,
    short iValidThreshold)
{
    extern int      in_buffer_select;
//  extern int      first_pass;
```

peak_isr.c continued

```
# if PKDT_DEBUG
    int i;
    for (i = 0 ; i < PKDT_CCD_MAX_LENGTH ; i++) {
        aiCCD_filter_signal[i] = aiCCD_derive_1st[i] = 0;
        aiCCD_derive_2nd[i] = aiDebug [i] = 0;
    }
# endif

    in_buffer_select = 0;
    first_pass = TRUE;
    iThreshold = iValidThreshold;

    iSkipFrom = iCCDWindowFrom - FIR_D1_ORDER / 2;
    if (iSkipFrom < 0) iSkipFrom = 0;
    iLength = iCCDWindowLength + (FIR_D1_ORDER + FIR_D2_ORDER);
    iSkipEnd = iCCDLength - iLength - iSkipFrom;
    if (iSkipEnd < 0) {        /* Error */
            iSkipEnd = 0;
            iLength = iCCDLength - iSkipFrom;
            return 0;
    }
    else return 1;
}


/*
    iPeakDetect
        Returns the iNbPeak detected in the signal
*/

int iPeakDetect (pkdt_data_struct *ptPkDt, int const iMaxNbPeak)
{
    register int i,iPos, iInt;
    int iPeak;
    short  iValid;

    extern int in_buffer_select;

    /* READ THE CCD SIGNAL, OUTPUT into: Filtered Intensity buffer */

# if G_PKDT_INDATA_TYPE == 0 // FIFO mode, not buffered mode
    /* Flush the first iSkipFrom pixel in the CCD */
    for (i = iSkipFrom ; i-- > 0 ; ) aiSignal[0] = G_PKDT_INDATA_FIFO;
# endif

    /* Read the signal and filter the signal (Intensity) */
    for (i = 0 ; i < iLength ; i++)
    {
        if (in_buffer_select == 0)    // use buffer 0
        {
            # if G_PKDT_INDATA_TYPE == 1    // buffered
                    aiSignal[i & (FIR_INT_LENGTH - 1)] = G_PKDT_INDATA_BUFFER_0(i+iSkipFrom);
            # elif G_PKDT_INDATA_TYPE == 0
                    aiSignal[i & (FIR_INT_LENGTH - 1)] = G_PKDT_INDATA_FIFO;
            # endif
                    aiFilterSignal[i] = FIL_INTENSITY(aiSignal);
        }

        else    // use buffer 1
        {
            # if G_PKDT_INDATA_TYPE == 1    // buffered
                    aiSignal[i & (FIR_INT_LENGTH - 1)] = G_PKDT_INDATA_BUFFER_1(i+iSkipFrom);
            # elif G_PKDT_INDATA_TYPE == 0
```

peak_isr.c continued

```
                aiSignal[i & (FIR_INT_LENGTH - 1)] = G_PKDT_INDATA_FIFO;
            #endif
                aiFilterSignal[i] = FIL_INTENSITY(aiSignal);
        }
    }

#if G_PKDT_INDATA_TYPE == 0
    /* Flush the remaining iSkipTo Pixel in the CCD */
    /* Note correction: iSkipEnd not iSkipFrom, d.g. 99-12-14 */
    for (i = iSkipEnd ; i-- > 0 ; ) aiSignal[0] = G_PKDT_INDATA_FIFO;
#endif

    /* Continue with peak detection */
    /* FIRST DERIVATIVE ON INTENSITY FILTER
        [1 1 1 1][1 0 0 0 0 -1] => [1 1 1 1 0 -1 -1 -1 -1] or
        [1 1][1 0 0 -1] => [1 1 0 -1 -1]

    */

    for (i = FIR_D1_TAP ; i < iLength ; i++) {
        aiDerive1st[i] = aiFilterSignal[i] - aiFilterSignal[i-FIR_D1_TAP];
#if PKDT_DEBUG
        aiCCD_filter_signal[i] = aiFilterSignal[i];
        aiCCD_derive_1st[i] = aiDerive1st[i];
#endif
    }

    /* Validate 1st derivative */
    /* Check for zero-crossing, detect peak positions */
    iPeak = 0;
    for (i = FIR_D1_TAP + FIR_D2_TAP ; i < iLength ; i++) {
        /* SECOND DERIVATIVE SIGNAL */
        iValid = FIL_DERIVE(aiDerive1st) + iThreshold;   /* 1999-12-14, d.g. */


/***** Begin Testing ********/
aiCCD_derive_2nd[i] = iValid;
/***** End Testing ********/


#if PKDT_DEBUG
        aiCCD_derive_2nd[i] = iValid;
        aiDebug [i] |= 1;
#endif

        if (iValid < 0) {
            iPos = i - FIR_D2_ORDER / 2;        /* Delay introduced by 2nd FIR */
#if PKDT_DEBUG
            aiDebug [iPos] |= 2;
#endif
            if ((aiDerive1st[iPos] >= 0) && (aiDerive1st[iPos+1] < 0)) {
                if (iPeak < iMaxNbPeak) {
                        /* Compensate for the window and processing */
                        /* Delay introduce by 1st FIR - Intensity filtering section */
                        /* Truncation /2 important */
                    iInt = iPos - (FIR_D1_ORDER/2 - FIR_INT_LENGTH/2);

                    // [2001-03-22, D.G.]
                    //  When 'short' declarations are used, caste the following interpolation
                    //  as 'ints' to reduce truncation effects. Failure to do so produces
                    //  noticeable banding in the resulting image.
                    ptPkDt[iPeak].lPeakPosQ6 =      /* Add filtering */
                        ((iInt - FIR_INT_LENGTH/2 + iSkipFrom) << PKDT_PIX_QFORMAT)
                        + ((int)aiDerive1st[iPos] << PKDT_PIX_QFORMAT) /
```

peak_isr.c continued

```
                    ((int)aiDerive1st[iPos] - (int)aiDerive1st[iPos+1]);

# if PKDT_DEBUG
                    aiDebug [iPos] |= 4;
                    aiDebug [iInt] |= 8;
# endif
                    ptPkDt[iPeak].lPeakInt = aiFilterSignal[iInt];  /* Filter gain */
                    ptPkDt[iPeak].lPeakSigma = 0;      /* Or .... */
                    ptPkDt[iPeak].lNotUsed = 0;
                    iPeak++;
                }
            }
        }
    }
    /* Clear remainder of structure */
    for (i = iPeak ; i < iMaxNbPeak ; i++) {
        ptPkDt[i].lPeakPosQ6 = 0;
        ptPkDt[i].lPeakInt = ptPkDt[i].lPeakSigma = ptPkDt[i].lNotUsed = 0;
    }

    return iPeak;          /* Total number of peaks */
}
```

iim62.cmd

```
/* TMS320C6201 SYSTEM MEMORY MAP for M62*/

/* MAP 0 */
/*
MEMORY
{
   SBSRAM:   o = 00000000h  l = 00040000h   Kernel data
   IPM:      o = 01400000h  l = 00010000h
   INT:      o = 02000000h  l = 00000200h   Interrupt vectors
   SDRAM0:   o = 02000200h  l = 003FFE00h
   SDRAM1:   o = 03000000h  l = 00400000h
   IDM:      o = 80000000h  l = 00010000h
}
*/

/* MAP 1 - modified for IIM62 */
MEMORY
{
   IPM:      o = 00000000h  l = 00010000h    /* program cache */
   IDM:      o = 80000000h  l = 00010000h    /* intenal data memory */
   INT:      o = 02000000h  l = 00000200h    /* Interrupt vectors */
   SDRAM0:   o = 02000200h  l = 00fffe00h    /* 16MB - (int vecs) */
/*  SDRAM0:  o = 02000200h  l = 00fbfe00h */  /* 16MB - (int vecs + Kernel data) */
/*  SDRAM1:  o = 02fc0000h   l = 00040000h */ /* 256k kernel data */
    ASRAM    o = 01600000h  l = 00080000h    /* 128k x 32 (512kB) For PCI interface */
}

/*
With the exception of .text, the initialized and uninitialized sections cannot
be allocated into internal program memory.
Initialized sections:
.cinit, .const, .switch section, and .text
Uninitialized sections:
.bss section, .far section, .stack section, .sysmem
*/

/* Start SPR P153-059-01 */

SECTIONS
/*
{     for testing, put all sections in onchip memory
   .text      >     IPM
   .vectors   >     IPM      Must be in same section as .text
   .stack     >     IDM
   .bss       >     IDM
   .cinit     >     IDM
   .cio       >     IDM
   .const     >     IDM
   .data      >     IDM
   .switch    >     IDM
   .sysmem    >     IDM
   .far       >     IDM
   .ipmtext   >     IDM
   .vec       >     IDM
}
*/
```

iim62.cmd continued

```
{   /* IPM is reserved for program cache */
   .text        >      SDRAM0
    .vectors    >      INT    /* Must be in same section as .text */
   .stack       >      IDM
   .bss         >      IDM
/*    .cinit    >      IDM */
   .cinit       >      SDRAM0
   .cio         >      IDM
   .const       >      IDM
   .data        >      IDM
   .switch      >      IDM
   .sysmem      >      SDRAM0
/* .far        >      IDM     */
/* .ipmtext    >      IDM     */
/* .vec        >      IDM     */
   .far         >      SDRAM0
   .ipmtext     >      SDRAM0
   .vec         >      SDRAM0
}
```

peak5.mki

```
#       gen62.mki
#   This .mki file is for the c6201 processor.
#
#

#
# Macros
#
ASM = asm6x
ASM_ARGS = -s -e
CC = cl6x
# CC_ARGS = -g -q -x2 -o2 -me -ml2
CC_ARGS = -q -mgx -o3 -op2 -pm -me -ml2 -mv6201
LNK = lnk6x
LNK_ARGS = -c -x -stack 0x800 -heap 0x0800000
AR = ar6x
AR_ARGS = -r
HEX = hex6x
HEX_ARGS = prom.cmd
HEX2BIN = hex2bin
HEX2BIN_ARGS = -f
HEXVERT = hexvert
HEXVERT_ARGS = -w4 -r
LIBS = -l stdio.lib -l periph.lib -l user.lib -l dsp.lib -l rts6201le.lib -l iim62.lib -l mqx.lib
DEBUGGER = c:\composer\cc_app.exe
# EXECUTE = terminal.exe

# Uncomment to explicitly set output object file
# OUTPUT = filename.out
```

code62A.c

```
/* Galvo M62 Selector file */
#include "c:\m6x\include\target\periph.h"
#include "c:\m6x\include\target\stdio.h"
#include "c:\m6x\include\target\user.h"

/* application includes */
#include "c:\Working\WaveformDemos\Demo5\galvo5\iim62\template.h"
#include "c:\Working\WaveformDemos\Demo5\galvo5\iim62\demodefs.h"
#include "c:\Working\WaveformDemos\Demo5\galvo5\iim62\galvo.h"
#include "c:\Working\WaveformDemos\Demo5\galvo5\iim62\externs.c"
#include "c:\Working\WaveformDemos\Demo5\galvo5\iim62\buildlut.c"
#include "c:\Working\WaveformDemos\Demo5\galvo5\iim62\compnext.c"
#include "c:\Working\WaveformDemos\Demo5\galvo5\iim62\main.c"
#include "c:\Working\WaveformDemos\Demo5\galvo5\iim62\galvo_isr.c"

/*-----------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------*/
```

template.h

```
// Prototypes
#pragma INTERRUPT(Galvo_isr);
void Galvo_isr(void);

/*-----------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------*/
```

demodefs.h

```
#ifndef  _demo_defs_h_
#define  _demo_defs_h_

#define full_version 0    // #ifdef used for early development

/* Global indicies: task template numbers. */
#define IPC_TTN                    10
#define USER_INTERFACE_TASK        11
#define GALVO_CONTROL_TASK         15
#define PEAK_DATA_TASK             20
#define FILE_HANDLER_TASK          25
#define GALVO_DATA_TASK            30

/* Queues - <#1 - 7 are reserved> */
#define USER_INTERFACE_Q           40
#define GALVO_CONTROL_Q            41
#define GALVO_DATA_Q               42
#define PEAK_DATA_Q                43
#define FILE_HANDLER_Q             44


/* Interprocessor message queues */
/* Q67 FIFOLinks - Board 1 */
// ProcA
#define   FLINK1_A_B_Q      10
#define   FLINK1_A_C_Q      11
#define   FLINK1_A_D_Q      12
// ProcB
#define   FLINK1_B_A_Q      13
#define   FLINK1_B_C_Q      14
#define   FLINK1_B_D_Q      15
// ProcC
#define   FLINK1_C_A_Q      16
#define   FLINK1_C_B_Q      17
#define   FLINK1_C_D_Q      18
// ProcD
#define   FLINK1_D_A_Q      19
#define   FLINK1_D_B_Q      20
#define   FLINK1_D_C_Q      21

/* Q67 FIFOLinks - Board 2 (future expansion) */
// ProcA
#define   FLINK2_A_B_Q      22
#define   FLINK2_A_C_Q      23
#define   FLINK2_A_D_Q      24
// ProcB
#define   FLINK2_B_A_Q      25
#define   FLINK2_B_C_Q      26
#define   FLINK2_B_D_Q      27
// ProcC
#define   FLINK2_C_A_Q      28
#define   FLINK2_C_B_Q      29
#define   FLINK2_C_D_Q      30
// ProcD
#define   FLINK2_D_A_Q      31
#define   FLINK2_D_B_Q      32
#define   FLINK2_D_C_Q      33

/* Processor numbers */
/* Q67 Board 1 */
#define Q67_1A_ID        1
#define Q67_1B_ID        2
#define Q67_1C_ID        3
#define Q67_1D_ID        4
```

demodefs.h continued

```
/* Q67 Board 2 */
#define Q67_2A_ID          5
#define Q67_2B_ID          6
#define Q67_2C_ID          7
#define Q67_2D_ID          8

/* M62 Boards (NOT REQUIRED) */
#define M62_1_ID           101    // galvo processor
#define M62_2_ID           102    // future expansion

/* misc */
#define    On_This_Processor   0
#define MAX_BUFFERS            16

/* DMA related */
#define DMA_CH_3               3 // DMA channel for FIFOPort access
#define BLOCK                  1
#define    NO_BLOCK            0

/* Galvo command types */
#define    INIT_GALVO          0x1122
#define STOP_GALVO             0x1010
#define START_GALVO            0x2020
#define DEFINE_WAVE            0x3030
#define NEW_AMP                0x4040
#define NEW_PHASE              0x5050
#define NEW_OFFSET             0x6060
#define CHANGE_WAVEFORM        0x7070
#define TEST_CASE              0x1080
#define PRINT_VALUES           0x1090

/* Galvo misc */
#define GALVO_ACK                  0x4040
#define GALVO_OK                   0x1111
#define BAD_GALVO_RESPONSE         0x2222
#define NO_GALVO_RESPONSE          0x3333
#define GALVO_NOT_READY            0x4444

/* Waveform types */
#define COS                    0
#define SAWTOOTH               1
#define UNDEFINED              2

/* Waveform Misc. */
#define MAX_NUM_WAVEFORMS      16
#define MAX_AXES               2
#define MAX_VOXELS_PER_LINE    0x1000        // 4096 voxels/line max
#define X_AXIS                 0
#define Y_AXIS                 1

/* Galvo run mode */
#define CONTINUOUS_MODE        0
#define IMAGE_MODE             0x1

/* Peak commands */
#define START_PEAK             0x5050
#define RQST_LINE              0x6060

/* Peak misc. */
#define INIT_FPORT             0x5555
#define PEAK_READY             0x1010
```

demodefs.h continued

```
/* No. of Peaks to Detect */
#define     NUMBER_OF_PEAKS      0x1

/* Peak Buffer information */
#define PEAK_BUFFER_SIZE         0x00800000  // 8 Mbytes = 2Mpeaks (@4 bytes/peak)
#define MAX_NUM_OF_VOXELS        PEAK_BUFFER_SIZE / (4 * NUMBER_OF_PEAKS) // (@ 4 bytes/peak)

/* FIFOPort information */
#define FIFOPORT_BUFFER_SIZE     0x200

/* File handler message types */
#define OPEN_FILE                0x0
#define WRITE_DATA               0x1
#define CLOSE_ALL_FILES          0x2

/* File handler data types */
#define POSITION                 0x0
#define INTENSITY                0x1

/* file status */
#define FILES_OPEN               0x1
#define FILES_CLOSED             0x0

/* Utility message Data types */
#define NO_WRITE                 0x1
#define OPEN_FAIL                0x2

#endif
/*-----------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------------*/
```

galvo.h

```
/*
 *  Rev        Date          By        Comments
 * ---------------------------------------------------------------------
 *  1.0        2001-01-31    d.g.      Initial release
 * ---------------------------------------------------------------------
 *
 *  Description:
 *             Header file for Galvo ISR.
 *
 */

/*---------------------------------------------------------------------
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *---------------------------------------------------------------------*/



#ifndef    __galvo_h__
#define    __galvo_h__

#define A4D1_SITE  0L
#define A4D4_SITE  0L

// DIO bits
#define DIO_PIN_1          0x0
#define DIO_PIN_2          0x1
#define DIO_PIN_3          0x2
#define DIO_PIN_7          0x6
#define X_SYNC             0x3
#define Y_SYNC             0x4
#define FIFO_EN            0x5

// DMA info
#define DMA_CH_0           0
#define BLOCK              1
#define NO_BLOCK           0

// Galvo State
#define    STOPPED         0x0
#define RUNNING            0x1
#define PENDING_STOP       0x2
#define PENDING_START      0x3
#define PENDING_CHANGE     0x4

// ADC conversion constant
// +/- 10 V, 16-bit ADC => 20 V / 2^16 = 305.18 uV/bit
#define ADC_CONVERSION_CONSTANT    0.00030518


typedef volatile struct scan_info_struct
{
    int          WAVE_TYPE;          // cos, sawtooth
    float        AMP;                // amplitude in volts
    float        OFFSET;             // offset in volts
    float        PHASE;              // radians
    float        PHASE_CORRECTION;   // radians
    short        VOXELS_PER_LINE;
    short        CYCLES;             // cycles per line (lissajous)
    short        RASTER_INC;         // incr for this axis, triggered by other axis
    short        RESERVED;           // for quadbyte alignment
} SCAN_INFO_STRUCT, * SCAN_INFO_STRUCT_PTR;
```

galvo.h continued

```
typedef volatile struct waveform_define_struct
{
    SCAN_INFO_STRUCT     SCAN_INFO_X;    // x-axis waveform
    SCAN_INFO_STRUCT     SCAN_INFO_Y;    // y-axis waveform
} WAVEFORM_DEFINE_STRUCT, * WAVEFORM_DEFINE_STRUCT_PTR;


typedef volatile struct    active_wave_state_struct
{
    short          X_VOXEL_CNTR;
    short          Y_VOXEL_CNTR;

    short          X_DAC_VALUE;
    short          Y_DAC_VALUE;

    short          STATE;
    short          RESERVED;
} ACTIVE_WAVE_STATE_STRUCT, * ACTIVE_WAVE_STATE_STRUCT_PTR;


typedef    volatile struct galvo_info_struct
{
    short                      ACTIVE_WAVE_INDEX;
    short                      EDIT_WAVE_INDEX;
    short                      PENDING_WAVE_INDEX;
    short                      RESERVED;

    ACTIVE_WAVE_STATE_STRUCT ACTIVE_WAVE_STATE;
    WAVEFORM_DEFINE_STRUCT        WAVEFORM_DEF[MAX_NUM_WAVEFORMS];
} GALVO_INFO_STRUCT, * GALVO_INFO_STRUCT_PTR;


// prototypes
void build_luts( GALVO_INFO_STRUCT_PTR );
void compute_next_sample( GALVO_INFO_STRUCT_PTR );

#endif /* Galvo_h */
```

externs.c

```
/*
 *  Rev        Date          By        Comments
 * ---------------------------------------------------------------------
 *  1.0        2001-01-31    d.g.      Initial release
 * ---------------------------------------------------------------------
 *
 *  Description:
 *
 *             external declarations
 *
 */


 GALVO_INFO_STRUCT_PTRgalvo_info_ptr;

 // lookup table.

#pragma DATA_SECTION (wave_lut, ".sysmem");
float       wave_lut[MAX_NUM_WAVEFORMS] [MAX_AXES] [MAX_VOXELS_PER_LINE];

/*-------------------------------------------------------------------
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *-------------------------------------------------------------------*/
```

buildlut.c

```
/*
 * build_luts
 *
 *  FILENAME:        \buildlut.c
 *
 *  PARAMETERS:      galvo_info_struct_ptr
 *
 *  DESCRIPTION:     This function pre-computes a pair of lookup tables based on
 *                   information contained in the galvo_info_struct. This
 *                   is done for performance reasons. The tables associate
 *                   an index with a cos() or sawtooth value. Phase, amplitude
 *                   and offset adjustments must be computed later.
 *
 *                   THIS VERSION ONLY GENERATES 1 LUT
 *
 *
 *  RETURNS:         none
 *
 *  AUTHOR:          D.Green
 *
 *  HISTORY:
 *          Date            Who       What
 *          2001-04-27      D.G.      first version -  1D cos only
 *          2001-05-16      D.G       2 axes, multiple waveforms
 *          2001-05-17      D.G.      sawtooth
 */

#include "c:\ti\c6000\cgtools\include\math.h"

void build_luts( GALVO_INFO_STRUCT_PTR galvo_info_struct_ptr )
 {
    float const     _2_pi = 6.283185;
    short           x_cycles;
    short           y_cycles;
    int             i;
    short           line_size;
    short           wave_num;
    int             x_wave_type;
    int             y_wave_type;
    short           x_line_size;
    short           y_line_size;
    extern float    wave_lut[MAX_NUM_WAVEFORMS] [MAX_AXES] [MAX_VOXELS_PER_LINE];

    wave_num = galvo_info_struct_ptr->EDIT_WAVE_INDEX;
    x_wave_type = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.WAVE_TYPE;
    y_wave_type = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.WAVE_TYPE;

    x_cycles = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.CYCLES;
    y_cycles = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.CYCLES;
    line_size = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.VOXELS_PER_LINE;

    if ((x_wave_type == COS) & (y_wave_type == COS))
    {
        for (i = 0; i < line_size; ++i )
        {
            wave_lut[wave_num][X_AXIS][i] = cosf( _2_pi*i*x_cycles/line_size );
            wave_lut[wave_num][Y_AXIS][i] = cosf( _2_pi*i*y_cycles/line_size );
        } // Endfor
    }

    else if ((x_wave_type == SAWTOOTH) & (y_wave_type == SAWTOOTH))
    {
        x_line_size =
            galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.VOXELS_PER_LINE;
```

buildlut.c continued

```
        y_line_size =
            galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.VOXELS_PER_LINE;

        for (i = 0; i < x_line_size; ++i )
            wave_lut[wave_num][X_AXIS][i] = (2.*i/(x_line_size - 1)) -1.;

        for (i = 0; i < y_line_size; ++i )
            wave_lut[wave_num][Y_AXIS][i] = (2.*i/(y_line_size - 1)) -1.;

        return;
    }

    else
    {   // undefined case, build NULL tables.
        for (i = 0; i < line_size; ++i )
        {
            wave_lut[wave_num][X_AXIS][i] = 0;
            wave_lut[wave_num][Y_AXIS][i] = 0;
        }
    } // Endif

 } //Endbody

/*-----------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------------*/
```

compnext.c

```
/*
 * compute_next_sample()
 *
 *  FILENAME:     C:\Working\WaveformDemos\Demo1\Galvo1\iim62\compnext.c
 *
 *  PARAMETERS:     galvo_info_struct_ptr
 *
 *  DESCRIPTION:    This function computes the next pair of data samples
 *                  for 2 dimensional scanning waveform generation. It
 *                  computes both x and y values for raster and lissajous
 *                  waveforms. The argument points to a struct that fully
 *                  describes the waveforms and their current state.
 *                  Waveforms are computed as floats.
 *
 *                  DAC gain adjustment:
 *                  16-bit DAC with +/- 10 volt range.
 *                  i.e. 2^16/20 counts/volt = 3276.8 counts/volt.
 *                  The cosf() generates +/- 1 volt signal so the
 *                  following correction is required:
 *                  x_to_DAC = 3276.8 * cosf();
 *                  Amplitude, phase and offset corrections are also
 *                  required.
 *
 *  RETURNS:        none
 *
 *  AUTHOR:         D.Green
 *
 *  HISTORY:
 *          Date            Who     What
 *          2001-04-27      D.G.    first version -  1D cos only
 *          2001-05-16      D.G.    2-axes, multiple waveforms
 *          2001-05-17      D.G.    COS or sawtooth waveforms.
 *          2001-06-08      D.G.    generalized waveforms
 */

#include "c:\ti\c6000\cgtools\include\math.h"

void compute_next_sample( GALVO_INFO_STRUCT_PTR galvo_info_struct_ptr )
{ //Body

    float const     _2_pi = 6.283185;
    short           wave_num;
    short           x_cycles;
    float           x_amp;
    float           x_phase;
    float           x_phase_correction;
    float           x_offset;
    short           y_cycles;
    float           y_amp;
    float           y_phase;
    float           y_phase_correction;
    float           y_offset;
    int             ix, iy;
    int             x_line_size, y_line_size;
    short           x_val;
    int             x_index;
    short           y_val;
    int             y_index;
    extern float    wave_lut[MAX_NUM_WAVEFORMS][MAX_AXES][MAX_VOXELS_PER_LINE];

    wave_num = galvo_info_struct_ptr->ACTIVE_WAVE_INDEX;

    x_cycles = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.CYCLES;
    x_amp = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.AMP;
```

compnext.c continued

```
    x_phase =
        galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.PHASE;
    x_phase_correction =
        galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.PHASE_CORRECTION;
    x_offset = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.OFFSET;

    y_cycles = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.CYCLES;
    y_amp = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.AMP;
    y_phase = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.PHASE;
    y_phase_correction =
        galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.PHASE_CORRECTION;
    y_offset = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.OFFSET;

    x_line_size = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_X.VOXELS_PER_LINE;
    y_line_size = galvo_info_struct_ptr->WAVEFORM_DEF[wave_num].SCAN_INFO_Y.VOXELS_PER_LINE;
    ix = galvo_info_struct_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR;
    iy = galvo_info_struct_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR;

    // phase is in radians
    x_index = ix + ((x_phase + x_phase_correction)*x_line_size)/(_2_pi * x_cycles);
    x_index %= x_line_size; // wrap
    // Note gain adjustment for DACs. See note above.
    x_val = (short)(3276.8 * (x_amp * wave_lut[wave_num][X_AXIS][x_index] + x_offset));

    y_index = iy + ((y_phase + y_phase_correction)*y_line_size)/(_2_pi * y_cycles);
    y_index %= y_line_size; // wrap
    y_val = (short)(3276.8 * (y_amp * wave_lut[wave_num][Y_AXIS][y_index] + y_offset));

    galvo_info_struct_ptr->ACTIVE_WAVE_STATE.X_DAC_VALUE = x_val;
    galvo_info_struct_ptr->ACTIVE_WAVE_STATE.Y_DAC_VALUE = y_val;

    return;

} //Endbody

/*-----------------------------------------------------------------------
*  (C), Her Majesty the Queen in right of Canada
*  as represented by the National Research Council, Canada, 2001
*-----------------------------------------------------------------------*/
```

main.c

```
/*
 *  Rev    Date          By        Comments
 * ---------------------------------------------------------------------------
 *  1.0    2001-02-02    d.g.      Initial release
 *         2001-12-14    DG        now transfers galvo posn data via FPorts
 *                                  (major code restructuring)
 * ---------------------------------------------------------------------------
 *
 *  Description:
 *
 *              this does not run under MQX
 *              Test for Galvo isr.
 *
 *  Parameters:
 *
 *  Returns:
 *
 */

/*-----------------------------------------------------------------------
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *-----------------------------------------------------------------------*/
#include "c:\m6x\include\target\periph.h"
#include "c:\m6x\include\target\stdio.h"

void main()
{
    extern GALVO_INFO_STRUCT_PTR      galvo_info_ptr;
    int                               DONE;
    AIO_PAIR volatile                 out_pair[1];

    clrscr();

    DONE = FALSE;

    // initialize FIFOPort
    enable_fifo_port();
    reset_fifo_port();

    // allocate struct
    galvo_info_ptr = (GALVO_INFO_STRUCT_PTR)malloc( sizeof( GALVO_INFO_STRUCT ) );
    if ( galvo_info_ptr == 0 )
        printf("Galvo main(): _malloc() failed. Unable to allocate galvo_info struct.\n");


/*
    some DAC values

    volts   Code
    -----   ----
    +5      0x4000
    0       0x8000
    -5      0xC000
*/

    // initialize info_ptr
    galvo_info_ptr->ACTIVE_WAVE_INDEX = 0;
    galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR = 0;
    galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR = 0;
    galvo_info_ptr->ACTIVE_WAVE_STATE.X_DAC_VALUE = 0;
    galvo_info_ptr->ACTIVE_WAVE_STATE.Y_DAC_VALUE = 0;
    galvo_info_ptr->ACTIVE_WAVE_STATE.STATE = STOPPED;
```

main.c continued

```
    //  Setup M62 digital I/O direction MSBy...LSBy
    dig_dir(DIO_DIR_INPUT, DIO_DIR_INPUT, DIO_DIR_INPUT, DIO_DIR_OUTPUT);
    write_dig_bit( FIFO_EN, 0 ); // negate FIFO_EN

    puts("\nGalvo5\n------\n\nGalvo Initialization. \n");
    puts("Dig I/O pin 7 (marker) is asserted for the duration of the isr.\n");
    printf("galvo_info_ptr = 0x%x\n", galvo_info_ptr);

    A4D4_trigger_dac_pair(A4D4_SITE, 0, SOFTWARE); // setup DAC Trigger source.
    A4D4_trigger_adc_pair(A4D4_SITE, 0, SOFTWARE); // setup ADC Trigger source.

    // initialize dacs to 0 volts.
    out_pair[0].half.low = galvo_info_ptr->ACTIVE_WAVE_STATE.X_DAC_VALUE;
    out_pair[0].half.high = galvo_info_ptr->ACTIVE_WAVE_STATE.Y_DAC_VALUE;
    A4D4_write_dac_pair(A4D4_SITE, 0, out_pair[0].both); // update buffer
    A4D4_convert_dac_pair(A4D4_SITE, 0); // convert

    EnableInterrupts();

    // 2001-12-13
    // initialize Q67's FIFOPort threshold via handshake
    // do this BEFORE activating voxel clock interrupts.
    enable_clock();
    while (!DONE)
    {
        // This loop checks for an INIT_FPORT command from the Galvo_Data_Task().

        ms(1); // awake every millisecond
        if ( (get_fifo_port_status() & Rx_FIFO_EMPTY) == 1 ) // if (not empty)...
        {
            switch (fifo_port_eat())
            {
                case INIT_FPORT:
                { // set Q67 FPort threshold to Half full condition.
                    set_fifo_port_AF_levels( 0, 252 );
                    DONE = TRUE;
                    //ClearInterrupt(EINT2_INTERRUPT);

                    // [2001-08-30] send ack to Q67
                    fifo_port_put16(PEAK_READY);
                    break;
                } /* Endcase */

                default:
                    break;

            } /* Endswitch */
        } /* Endif */
    } /* Endwhile */


    // Init complete, now setup Interrupt handling for EINT2_INTERRUPT
   InterruptSource( EINT2_INTERRUPT, EINT2_INT_SRC );
   ClearInterrupt(EINT2_INTERRUPT);
   ActivateInterrupt(EINT2_INTERRUPT);
    InstallIntVector(Galvo_isr, EINT2_INTERRUPT);
   EnableInterrupt(EINT2_INTERRUPT);

    //loop forever
    while(TRUE);
}
```

galvo_isr.c

```
/*
 *  FUNCTION: Galvo_isr
 *
 *  PARAMETERS:
 *
 *  RETURNS:
 *
 *  DESCRIPTION:
 *              Galvo control Interrupt Service Routine
 *              Triggered by Voxel Clock.
 *
 *              This version accommodates full waveform generation.
 *              The waveform definition struct is passed from the
 *              Galvo_control_task using a fifoport dma.
 *              Waveforms can be dynamically selected at runtime, and
 *              selection occurs synchronously at the end a line.
 *
 *              Command dma transfers are NOT acknowledged in this version.
 *
 *              Raster scans are generalized to be either vertical or horizontal.
 *              Raster terminology used here:
 *                  - fast axis is the 'primary' axis,
 *                  - slow axis is the 'secondary' axis
 *              The secondary axis must be enabled for raster generation.
 *
 *                              Waveform Generation Table
 *              ---------------------------------------------------------------------
 *              Waveform      x_Vox_inc     y_Vox_inc     x_raster_inc  y_raster_inc
 *              ---------------------------------------------------------------------
 *              Lissajous     x_cycles      y_cycles          0             0
 *              Vector        1                 1             0             0
 *              Raster-x      1                 0             0             1
 *              Raster-y      0                 1             1             0
 *
 *              e.g. For a raster with x-axis the primary (fast) axis, use
 *              x_raster_inc = 0, y_raster_inc = 1
 *
 *
 *
 *  AUTHOR:       D. Green
 *
 *  HISTORY:
 *              Date          by      Desc.
 *              ----          --      -----
 *              2001-04-26    D.G.    first version cos() on the fly.
 *              2001-04-27    D.G.    uses cos() LUT for performance
 *              2001-05-16    D.G.    2 axis, multiple waveform definitions
 *              2001-06-08    D.G.    generalized waveforms
 *              2001-06-29    D.G.    fixed bug: added sawtooth slow axis wrap
 *              2001-11-27    D.G.    raw galvo position acquisition and transfer
 *
 *
 *  (C), Her Majesty the Queen in right of Canada
 *  as represented by the National Research Council, Canada, 2001
 *
 */

void Galvo_isr (void)
{ //Body

    extern GALVO_INFO_STRUCT_PTR      galvo_info_ptr;
    AIO_PAIR volatile                 in_pair[1], out_pair[1];
    short                             x_sync;
    short                             galvo_command;
```

galvo_isr.c continued

```
short               state;
short               wave_index;
short               edit_index;
short               transfer_count;
int                 end_of_line;

int                 x_val, y_val;
float               * x_flt_ptr;
float               * y_flt_ptr;
const float         _RAD_TO_DEG = 57.29578;

ClearInterrupt(EINT2_INTERRUPT);

// assert pin 7 - ISR duration marker
write_dig_bit( DIO_PIN_7, 1 );

end_of_line = FALSE;
state = galvo_info_ptr->ACTIVE_WAVE_STATE.STATE;
wave_index = galvo_info_ptr->ACTIVE_WAVE_INDEX;

// Check for new command from Galvo Control Task
if ( (get_fifo_port_status() & Rx_FIFO_EMPTY) == 1 ) // if (not empty)...
{
    galvo_command = fifo_port_eat();

    switch (galvo_command)
    {
        case INIT_GALVO:
        {
            // set state and exit
            galvo_info_ptr->ACTIVE_WAVE_STATE.STATE = STOPPED;

            // negate pin 7 - ISR duration marker
            write_dig_bit( DIO_PIN_7, 0 );
            ClearInterrupt(EINT2_INTERRUPT);
            return;
        } // Endcase

        case START_GALVO:
        {
            // initialize state, compute first samples, then exit
            state = PENDING_START;
            galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR = 0;
            galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR = 0;

            galvo_info_ptr->ACTIVE_WAVE_INDEX = fifo_port_get16(); // get WAVE_NUM

            // compute first galvo samples
            compute_next_sample( galvo_info_ptr ); // generate initial sample pair

            galvo_info_ptr->ACTIVE_WAVE_STATE.STATE = state;

            // negate pin 7 - ISR duration marker
            write_dig_bit( DIO_PIN_7, 0 );
            ClearInterrupt(EINT2_INTERRUPT);
            return;
        } // Endcase

        case STOP_GALVO:
        {   // update state
            if ( state != STOPPED ) // already stopped?
                state = PENDING_STOP;
            break;
        } // Endcase
```

galvo_isr.c continued

```
case CHANGE_WAVEFORM:
{   // if running, switch to new waveform at end of current line.
    // else leave stopped.
    if (state != STOPPED)
        state = PENDING_CHANGE;
    galvo_info_ptr->PENDING_WAVE_INDEX = fifo_port_get16(); // (busywaits)
    break;
} // Endcase

case DEFINE_WAVE:
{
    // Rebuild the specified waveform struct.
    // This cannot be done concurrently with waveform generation or
    // else interrupts will be dropped. Upper level code must monitor the state
    // since no command acks are available.
    galvo_info_ptr->EDIT_WAVE_INDEX = edit_index = fifo_port_eat();
    transfer_count = fifo_port_eat(); // (does not busywait)
    fifo_port_eat(); // flush

    dma16_from_fifo(
        DMA_CH_0,                                          // DMA ch 0
        (short*)FIFOPORT_BASE,
        (short*)&galvo_info_ptr->WAVEFORM_DEF[edit_index], // destination
        (int)transfer_count - 3,
        BLOCK);                                            // wait for completion

    // build LUTs associated with this new waveform
    build_luts( galvo_info_ptr );

    break;
} // Endcase

case NEW_AMP:
{
    // coerce the int to a float via pointers:
    x_val = fifo_port_get32();
    y_val = fifo_port_get32();
    x_flt_ptr = (float *)&x_val;
    y_flt_ptr = (float *)&y_val;
    galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.AMP = * x_flt_ptr;
    galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.AMP = * y_flt_ptr;
    break;
} // Endcase

case NEW_PHASE:
{
    x_val = fifo_port_get32();
    y_val = fifo_port_get32();
    x_flt_ptr = (float *)&x_val;
    y_flt_ptr = (float *)&y_val;
    galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.PHASE = * x_flt_ptr;
    galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.PHASE = * y_flt_ptr;
    break;
} // Endcase

case NEW_OFFSET:
{
    x_val = fifo_port_get32();
    y_val = fifo_port_get32();
    x_flt_ptr = (float *)&x_val;
    y_flt_ptr = (float *)&y_val;
    galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.OFFSET = * x_flt_ptr;
    galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.OFFSET = * y_flt_ptr;
```

galvo_isr.c continued

```
            break;
        }

        case PRINT_VALUES:
        {
            // for debugging, prints to M62 Stdio:
            clrscr();

            if (galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.WAVE_TYPE == COS)
                puts("x_type = COS\t\t");
            else
                puts("x_type = SAWTOOTH\t\t");

            if (galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.WAVE_TYPE == COS)
                puts("y_type = COS\n");
            else
                puts("y_type = SAWTOOTH\n");

            printf("x_amp = % 5.1f\t\t\t y_amp = % 5.1f\n",
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.AMP,
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.AMP);

            printf("x_offset = % 5.1f\t\t y_offset = % 5.1f\n",
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.OFFSET,
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.OFFSET);

            printf("x_phase(deg) = % 5.1f\t\tx_phase_corr(deg) = % 5.1f\n",
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.PHASE * _RAD_TO_DEG,
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.PHASE_CORRECTION
                    * _RAD_TO_DEG);

            printf("y_phase(deg) = % 5.1f\t\ty_phase_corr(deg) = % 5.1f\n",
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.PHASE * _RAD_TO_DEG,
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.PHASE_CORRECTION
                    * _RAD_TO_DEG);

            printf("x_vox_per_line = %d\ty_vox_per_line = %d\n",
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.VOXELS_PER_LINE,
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.VOXELS_PER_LINE);

            printf("x_cycles = %d\t\t\ty_cycles = %d\n",
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.CYCLES,
                galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.CYCLES);

            break;
        }

        default: // unknown command
            break;

    } // Endswitch
} // Endif

if (state == PENDING_START)
{
    // enable FIFO clock
    write_dig_bit( FIFO_EN, 1 ); // assert DIO pin 6, FIFO_EN
    state = RUNNING;
}

if ( (state == RUNNING)
    || (state == PENDING_STOP)
    || (state == PENDING_CHANGE) )
{
```

galvo_isr.c continued

```
out_pair[0].half.low = galvo_info_ptr->ACTIVE_WAVE_STATE.X_DAC_VALUE;
out_pair[0].half.high = galvo_info_ptr->ACTIVE_WAVE_STATE.Y_DAC_VALUE;

// re-select trigger sources, THIS IS ESSENTIAL!!
A4D4_trigger_dac_pair(A4D4_SITE, 0, SOFTWARE);
A4D4_trigger_adc_pair(A4D4_SITE, 0, SOFTWARE);

// output next Galvo DAC sample pair
A4D4_write_dac_pair(A4D4_SITE, 0, out_pair[0].both); // update buffer
A4D4_convert_dac_pair(A4D4_SITE, 0); // convert

// read current galvo positions
A4D4_convert_adc_pair(A4D4_SITE, 0);
in_pair[0].both = A4D4_read_adc_pair(A4D4_SITE, 0);

// send galvo positions to Q67 via FIFOport as 16-bit ints
// conversion to float will occur in Galvo_Notifier().
fifo_port_spit( in_pair[0].half.low );
fifo_port_spit( in_pair[0].half.high );

// check to assert X_SYNC
if ( galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR == 0 )
{
    write_dig_bit( X_SYNC, 1 ); // assert DIO pin 4, X_SYNC
    x_sync = 1;
}

if ( x_sync == 1 )
{
    write_dig_bit( X_SYNC, 0 ); // negate DIO pin 4, X_SYNC
    x_sync = 0;
}

// Check for end of line.
if (galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.WAVE_TYPE != SAWTOOTH)
{   // Lissajous, Vector waveforms
    // incr both axes and wrap if necessary
    if (galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR++ >=
        (galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.VOXELS_PER_LINE - 1))
    {
        galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR = 0;
        end_of_line = TRUE;
    }

    if (galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR++ >=
        (galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.VOXELS_PER_LINE - 1))
    {
        galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR = 0;
        end_of_line = TRUE;
    }
}

else
{   // Sawtooth Waveforms
    if (galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.RASTER_INC)
    {   // incr y-axis, the primary (fast) axis.
        if (galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR++ >=
            (galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.VOXELS_PER_LINE - 1))
        {   // wrap
            galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR = 0;
            galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR += 1;
```

galvo_isr.c continued

```
                // wrap [2001-06-29]
                if (galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR >= (galvo_info_ptr->
                        WAVEFORM_DEF[wave_index].SCAN_INFO_X.VOXELS_PER_LINE - 1))
                    galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR = 0;

                end_of_line = TRUE;
            } // Endif
        }

        else if(galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_Y.RASTER_INC)
        {   // incr x-axis, the primary (fast) axis.
            if (galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR++ >=
                (galvo_info_ptr->WAVEFORM_DEF[wave_index].SCAN_INFO_X.VOXELS_PER_LINE - 1))
            {   // wrap
                galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR = 0;
                galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR += 1;

                // wrap [2001-06-29]
                if (galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR >= (galvo_info_ptr->
                        WAVEFORM_DEF[wave_index].SCAN_INFO_Y.VOXELS_PER_LINE - 1))
                    galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR = 0;

                end_of_line = TRUE;
            } // Endif
        } // Endif
    } // Endif

    if (end_of_line)
    {
        end_of_line = FALSE;

        if ( state == PENDING_STOP )
        {
            write_dig_bit( FIFO_EN, 0 ); // negate FIFO_EN
            state = STOPPED;
        } // Endif

        if (state == PENDING_CHANGE)
        {   // switch waveform now, at end of line.
            state = RUNNING;
            galvo_info_ptr->ACTIVE_WAVE_INDEX = galvo_info_ptr->PENDING_WAVE_INDEX;
            // clear the counters
            galvo_info_ptr->ACTIVE_WAVE_STATE.X_VOXEL_CNTR = 0;
            galvo_info_ptr->ACTIVE_WAVE_STATE.Y_VOXEL_CNTR = 0;
        } // Endif

    } // Endif

    // update waveforms, save state
    compute_next_sample(galvo_info_ptr);
    galvo_info_ptr->ACTIVE_WAVE_STATE.STATE = state;

} // Endif


// negate pin 7 - ISR duration marker
write_dig_bit( DIO_PIN_7, 0 );

} //Endbody
```

galvo5.mki

```
#   gen62.mki
#   This .mki file is for the c6201 processor.
#
#

#
# Macros
#
ASM = asm6x
ASM_ARGS = -s -e
CC = cl6x
CC_ARGS = -g -q -x2 -o2 -me -ml2
# CC_ARGS = -q -op2 -mg -mx -pm -o3 -me -ml2 -mv6201
LNK = lnk6x
LNK_ARGS = -c -x -stack 0x800 -heap 0x0800000
AR = ar6x
AR_ARGS = -r
HEX = hex6x
HEX_ARGS = prom.cmd
HEX2BIN = hex2bin
HEX2BIN_ARGS = -f
HEXVERT = hexvert
HEXVERT_ARGS = -w4 -r
LIBS = -l stdio.lib -l periph.lib -l user.lib -l dsp.lib -l rts6201le.lib -l iim62.lib -l mqx.lib
DEBUGGER = c:\composer\cc_app.exe
# EXECUTE = terminal.exe

# Uncomment to explicitly set output object file
# OUTPUT = filename.out
```

Appendix C. The User Interface Software.

This appendix lists the Visual Basic programs that provide the PC-based user interface. They are listed by form and module, starting with the main user interface window.

Main.frm

```
Private Sub cmdClear_Click()
   lblDisplay.Caption = ""
   char_buffer = ""
End Sub

Private Sub cmdGo_Click()
   Dim reply As Long
   Dim x_amp, y_amp As Integer

   SCANNING = True

   lblCmd.Caption = ""
   reply = Mailbox_send(WVFRM_GO)
   cmd_ack (reply)
End Sub

Private Sub cmdInit_Click()
'initialize target processors
   lblWait.Visible = True
   lblWait.Refresh
   Call DefineAllWaveformsCmd
   cmdInit.Enabled = False
   cmdLaunch.Enabled = False
   cmdQuit.Enabled = True
   cmdGo.Enabled = True
   cmdStop.Enabled = True
   cmdShowPins.Enabled = True
   lblFailed.Visible = False
   lblWait.Visible = False
   cmdStop_Click

End Sub

Private Sub cmdLaunch_Click()
   Dim reply As Long
reply = Mailbox_send(LAUNCH_TARGET)
   cmd_ack (reply)

   If reply = CMD_OK Then
      cmdGo.Enabled = False
      cmdStop.Enabled = False
      cmdLaunch.Enabled = False
      cmdInit.Enabled = True
      cmdShowPins.Enabled = False
   End If

End Sub

Private Sub cmdQuit_Click()
   Dim reply As Long

   reply = Mailbox_send(WVFRM_QUIT)
   cmd_ack (reply)
   End

End Sub

Private Sub cmdShowPins_Click()
   Dim reply As Long

   lblCmd.Caption = ""
   reply = Mailbox_send_no_ack(WVFRM_SHOW_PINS)
```

Main.frm continued

```
    cmd_ack (reply)
End Sub

Private Sub cmdStop_Click()
    Dim reply As Long

    lblCmd.Caption = ""
     reply = Mailbox_send_no_ack(WVFRM_STOP)

    cmd_ack (reply)
    SCANNING = False

End Sub

Private Sub dsp_OnInterrupt()
    Dim i, length As Long
    Dim int_length As Integer
    Dim point_index, value As Integer
    Dim accumulated_value, average_posn As Single
    Dim message_type As Integer

    Dim x_galvo_pos_raw, y_galvo_pos_raw As Integer

    Dim x_display, x_val(MAX_CHANGED_VOXELS_PER_LINE) As Single
    Dim x_sum, x_mean, x_dev, x_temp As Single

    Dim y_display, y_val(MAX_CHANGED_VOXELS_PER_LINE) As Single
    Dim y_sum, y_mean, y_dev, y_temp As Single

    On Error GoTo Error0:

    length = dsp.InterruptAcknowledge
    While dsp.MailInFull(0) = False 'wait till not empty
    Wend
    message_type = dsp.Mailbox(0)

    If message_type = STRING_TYPE Then
        If length = 0 Then
            Exit Sub
        End If

        For i = 0 To length - 1
            While dsp.MailInFull(0) = False
            Wend
            char_buffer = char_buffer & Chr(dsp.Mailbox(0))
        Next i

        lblDisplay.Caption = char_buffer

    ElseIf message_type = GRAPHIC_POINT_TYPE Then
        picDisplay.Cls 'clear the displays
        picBar.Cls
        If length = 0 Then  'exit on 0 length message
            Exit Sub
        End If

        x_line_size = WaveRegistry(WaveNum).Wave_X.WaveVoxPerLine
        If length > x_line_size Then
            msg = "Data count = " & Str(length)
            Title = "Message size error."
            MsgBox msg, 0, Title
            length = x_line_size
        End If
```

Main.frm continued

```
    accumulated_value = 0
    x_sum = 0
    y_sum = 0
    If length > 0 Then
       For i = 0 To length - 1
          point_index = dsp.Mailbox(0)    'busywait not used here...
          value = dsp.Mailbox(0) / 64
          accumulated_value = accumulated_value + Abs(value)
          x_galvo_pos_raw = dsp.Mailbox(0)
          y_galvo_pos_raw = dsp.Mailbox(0)

          'convert to volts and scale to match actual driven values:
          x_display = x_galvo_pos_raw * ADC_GAIN_CONSTANT * X_Galvo_Scale_Factor * Zoom
          y_display = y_galvo_pos_raw * ADC_GAIN_CONSTANT * Y_Galvo_Scale_Factor * Zoom
          x_val(i) = x_display
          y_val(i) = y_display
          picDisplay.PSet (x_display, y_display)
          x_sum = x_sum + x_display
          y_sum = y_sum + y_display
       Next i
    End If

     ' generate bar graph of "average range"
     If length > 0 Then
       average_posn = accumulated_value / length
      picBar.Line (0, 0)-(0, average_posn)
    End If

    ' generate centroid and standard deviation
    x_temp = 0
    y_temp = 0
    x_dev = 0
    y_dev = 0
    If length > 0 Then
       x_mean = x_sum / length
       y_mean = y_sum / length

       For i = 0 To length - 1
          x_temp = x_temp + Abs(x_val(i) - x_mean)
          y_temp = y_temp + Abs(y_val(i) - y_mean)
       Next i

       x_dev = 2 * (x_temp / (length))
       y_dev = 2 * (y_temp / (length))

    End If

    ' display box around centroid
    picDisplay.Line (x_mean - x_dev, y_mean + y_dev)-(x_mean + x_dev, y_mean - y_dev), RGB(0, 0,
255), B


   End If

   Exit Sub
Error0:
    msg = "Runtime error: " & Str(Err.Number)
    Title = "Fatal Error"
    MsgBox msg, 0, Title
End Sub
```

Main.frm continued

```
Private Sub Form_Load()

   cmdGo.Enabled = False
   cmdStop.Enabled = False
   cmdLaunch.Enabled = True
   cmdQuit.Enabled = False
   cmdInit.Enabled = False
   cmdShowPins.Enabled = False
   lblFailed.Visible = False
   lblWait.Visible = False

   Call Init_Waveforms
   WaveNum = 0

'  Setup Waveform display
   picDisplay.Scale (-2.5, 2.5)-(2.5, -2.5) 'coords of upper left and lower right corners
   picDisplay.DrawWidth = 1

'  Setup 'height' display
   picBar.Cls
   picBar.Scale (0, 100)-(1, 0)
   picBar.DrawWidth = 40

'  Initialize motion parameters
   MotionDirection = MOTION_DIR_BOTH
   MotionThreshold = MOTION_DIFF_THRESHOLD
   MotionFilterConstant = MOTION_K_FACTOR

'  Initialize Display scale factors
   X_Galvo_Scale_Factor = 2    'for now
   Y_Galvo_Scale_Factor = 1
   Zoom = 1

End Sub


Private Sub lblDisplay_Click()
   char_buffer = ""
   lblDisplay.Caption = ""
End Sub

Private Sub mnuAbout_Click()
   frmAbout.Show
End Sub

Private Sub mnuAdjust_Click()
   frmWvFrmEdit.Show vbModeless
End Sub

Private Sub mnuSelect_Click()

End Sub

Private Sub mnuDisplay_Click()
   frmDisplay.Show
End Sub

Private Sub mnuEdit_Click()
   frmSelect.Show vbModeless

End Sub

Private Sub mnuMotionEdit_Click()
```

Main.frm continued

```
End Sub

Private Sub mnuMotion_Click()
   frmMotionParams.Show vbModeless

End Sub

Private Sub picDisplay_Click()
   picDisplay.Cls

End Sub
```

frmSelect.frm

```
Private Sub cmdApply_Click()
'Update local registry and...
   Call UpdateWaveRegistry

'send command to update target
   reply = Mailbox_send(WVFRM_DEFINE)

   'check for command rejection
   If reply <> CMD_OK Then
      lblFailed.Visible = True
   Else
      lblFailed.Visible = False
   End If

   MBoxPrintf (Str(WaveNum)) 'Wave index

   'X axis:
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_X.WaveType)) 'X wavetype
   If WaveRegistry(WaveNum).Wave_X.WaveType = SAW Then
      If WaveRegistry(WaveNum).Wave_X.WaveRasterInc = 0 Then
         MBoxPrintf ("0") '0 if X is fast axis
      Else
         MBoxPrintf ("1") '1 if Y is fast axis
      End If
   End If

   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_X.WaveAmp))          'X Amplitude
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_X.WaveOffset))       'X Offset
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_X.WavePhase))        'X Phase
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_X.WavePhaseCorr))    'X Phase Correction
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_X.WaveVoxPerLine))   'X points
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_X.WaveCycles))       'X cycles

   'Y Axis:
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_Y.WaveType))         'Y wavetype
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_Y.WaveAmp))          'Y Amplitude
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_Y.WaveOffset))       'Y Offset
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_Y.WavePhase))        'Y Phase
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_Y.WavePhaseCorr))    'Y Phase Correction
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_Y.WaveVoxPerLine))   'Y points
   MBoxPrintf (Str(WaveRegistry(WaveNum).Wave_Y.WaveCycles))       'Y cycles


End Sub

Private Sub cmdClose_Click()
   Me.Hide

End Sub

Public Sub cmdSelect_Click()
   Dim x_amp, y_amp As Integer

   'send command to change selection
    reply = Mailbox_send_no_ack(WVFRM_SELECT)

   'send waveform index
   MBoxPrintf (Str(WaveNum))


    x_amp = WaveRegistry(WaveNum).Wave_X.WaveAmp
    y_amp = WaveRegistry(WaveNum).Wave_Y.WaveAmp

End Sub
```

frmSelect.frm continued

```
Private Sub Form_Activate()
   lblFailed.Visible = False
   optSel1(WaveNum).Enabled = True

   If SCANNING Then
      lblScanning.Visible = True
      cmdApply.Enabled = False
   Else
      lblScanning.Visible = False
      cmdApply.Enabled = True
   End If

   Call UpdateWaveDisplay

End Sub

Private Sub Form_Load()
   lblScanning.Visible = False
   cmdApply.Enabled = True

End Sub

Private Sub optSel1_Click(Index As Integer)
   WaveNum = Index
   Call UpdateWaveDisplay
End Sub
```

frmWvFrmEdit.frm

```
Private Sub cmdAmpApply_Click()
   Dim reply As Long
   Dim out_val As String

   'command
    reply = Mailbox_send_no_ack(WVFRM_EDIT_AMPLITUDE)

   'amplitude data
   out_val = Str(Format(WaveRegistry(WaveNum).Wave_X.WaveAmp, "###0.####"))
   MBoxPrintf (out_val)

   out_val = Str(Format(WaveRegistry(WaveNum).Wave_Y.WaveAmp, "###0.####"))
   MBoxPrintf (out_val)

End Sub


Private Sub cmdCancel_Click()
   Dim reply As Long

   'this action will terminate the edit mode
    reply = Mailbox_send_no_ack(WVFRM_ESCAPE_CMD)

   'close form
   frmWvFrmEdit.Hide

End Sub

Private Sub cmdOffApply_Click()
   Dim reply As Long
   Dim out_val As String

   WaveRegistry(WaveNum).Wave_X.WaveOffset = CSng(txtXOffset.Text)
   WaveRegistry(WaveNum).Wave_Y.WaveOffset = CSng(txtYOffset.Text)

   'command
    reply = Mailbox_send_no_ack(WVFRM_EDIT_OFFSET)

   'offset data
   On Error GoTo Err:
    out_val = Str(WaveRegistry(WaveNum).Wave_X.WaveOffset)
   MBoxPrintf (out_val)

    out_val = Str(WaveRegistry(WaveNum).Wave_Y.WaveOffset)
   MBoxPrintf (out_val)

' Catch all input formatting errors.
' this prevents the system from hanging
Err:
   out_val = "0."
   Resume Next

End Sub

Private Sub CmdPhaseApply_Click()
   Dim reply As Long
   Dim out_val As String

   WaveRegistry(WaveNum).Wave_X.WavePhase = txtXPhase.Text
   WaveRegistry(WaveNum).Wave_Y.WavePhase = txtYPhase.Text

   'command
    reply = Mailbox_send_no_ack(WVFRM_EDIT_PHASE)
```

frmWvFrmEdit.frm continued

```
    'phase data
    out_val = Str(Format(WaveRegistry(WaveNum).Wave_X.WavePhase, "##0.###"))
    MBoxPrintf (out_val)

    out_val = Str(Format(WaveRegistry(WaveNum).Wave_Y.WavePhase, "##0.###"))
    MBoxPrintf (out_val)
End Sub


Private Sub cmdZeroOffset_Click()
    'zero the scrollbars
    hsbXOffset.value = HSB_MAX_OFFSET / 2
    hsbYOffset.value = HSB_MAX_OFFSET / 2

End Sub

Private Sub cmdZeroPhase_Click()
    'zero the scrollbars
    hsbXPhase.value = HSB_MAX_PHASE / 2
    hsbYPhase.value = HSB_MAX_PHASE / 2

End Sub

Private Sub Form_Activate()
    Dim reply As Integer

    'initialize form
    txtWaveNum.Text = WaveNum

    'Amplitude
    hsbXAmp.Max = HSB_MAX_AMP
    hsbXAmp.Min = HSB_MIN_AMP
    hsbYAmp.Max = HSB_MAX_AMP
    hsbYAmp.Min = HSB_MIN_AMP

    hsbXAmp.value = ((WaveRegistry(WaveNum).Wave_X.WaveAmp / MAX_AMP) * HSB_MAX_AMP)
    txtXAmp.Text = WaveRegistry(WaveNum).Wave_X.WaveAmp

    hsbYAmp.value = ((WaveRegistry(WaveNum).Wave_Y.WaveAmp / MAX_AMP) * HSB_MAX_AMP)
    txtYAmp.Text = WaveRegistry(WaveNum).Wave_Y.WaveAmp

    'Phase
    hsbXPhase.Max = HSB_MAX_PHASE
    hsbXPhase.Min = HSB_MIN_PHASE
    hsbYPhase.Max = HSB_MAX_PHASE
    hsbYPhase.Min = HSB_MIN_PHASE

    hsbXPhase.value = ((WaveRegistry(WaveNum).Wave_X.WavePhase / MAX_PHASE) *
        HSB_MAX_PHASE / 2) + HSB_MAX_PHASE / 2
    txtXPhase.Text = Format(WaveRegistry(WaveNum).Wave_X.WavePhase, "##0.###")

    hsbYPhase.value = ((WaveRegistry(WaveNum).Wave_Y.WavePhase / MAX_PHASE) *
        HSB_MAX_PHASE / 2) + HSB_MAX_PHASE / 2
    txtYPhase.Text = Format(WaveRegistry(WaveNum).Wave_Y.WavePhase, "##0.###")

    'Offset
    hsbXOffset.Max = HSB_MAX_OFFSET
    hsbXOffset.Min = HSB_MIN_OFFSET
    hsbYOffset.Max = HSB_MAX_OFFSET
    hsbYOffset.Min = HSB_MIN_OFFSET

    hsbXOffset.value = ((WaveRegistry(WaveNum).Wave_X.WaveOffset / MAX_OFFSET) *
        HSB_MAX_OFFSET / 2) + HSB_MAX_OFFSET / 2
    txtXOffset.Text = Format(WaveRegistry(WaveNum).Wave_X.WaveOffset, "##0.###")
```

frmWvFrmEdit.frm continued

```
    hsbYOffset.value = ((WaveRegistry(WaveNum).Wave_Y.WaveOffset / MAX_OFFSET) *
        HSB_MAX_OFFSET / 2) + HSB_MAX_OFFSET / 2
    txtYOffset.Text = Format(WaveRegistry(WaveNum).Wave_Y.WaveOffset, "##0.###")

    'send WVFRM_EDIT command
    reply = Mailbox_send(WVFRM_EDIT)

    hsbXAmp.Enabled = True
    hsbYAmp.Enabled = True

End Sub

Private Sub hsbXAmp_Change()

    WaveRegistry(WaveNum).Wave_X.WaveAmp = (hsbXAmp.value / HSB_MAX_AMP) * MAX_AMP
    txtXAmp.Text = Format(WaveRegistry(WaveNum).Wave_X.WaveAmp, "##0.###")

End Sub

Private Sub hsbXOffset_Change()

    WaveRegistry(WaveNum).Wave_X.WaveOffset =
        ((hsbXOffset.value / HSB_MAX_OFFSET) - 0.5) * 2 * MAX_OFFSET
    txtXOffset.Text = Format(WaveRegistry(WaveNum).Wave_X.WaveOffset, "##0.###")
End Sub

Private Sub hsbXPhase_Change()
    WaveRegistry(WaveNum).Wave_X.WavePhase =
        ((hsbXPhase.value / HSB_MAX_PHASE) - 0.5) * 2 * MAX_PHASE
    txtXPhase.Text = Format(WaveRegistry(WaveNum).Wave_X.WavePhase, "##0.###")
End Sub

Private Sub hsbYAmp_Change()

    WaveRegistry(WaveNum).Wave_Y.WaveAmp = (hsbYAmp.value / HSB_MAX_AMP) * MAX_AMP
    txtYAmp.Text = Format(WaveRegistry(WaveNum).Wave_Y.WaveAmp, "##0.###")

End Sub

Private Sub hsbYOffset_Change()

    WaveRegistry(WaveNum).Wave_Y.WaveOffset =
        ((hsbYOffset.value / HSB_MAX_OFFSET) - 0.5) * 2 * MAX_OFFSET
    txtYOffset.Text = Format(WaveRegistry(WaveNum).Wave_Y.WaveOffset, "##0.###")
End Sub

Private Sub hsbYPhase_Change()
    WaveRegistry(WaveNum).Wave_Y.WavePhase =
        ((hsbYPhase.value / HSB_MAX_PHASE) - 0.5) * 2 * MAX_PHASE
    txtYPhase.Text = Format(WaveRegistry(WaveNum).Wave_Y.WavePhase, "##0.###")
End Sub
```

frmMotionParams.frm

```
Private Sub cmdApply_Click()

   On Error GoTo Error0:

   'fetch values

   If optDirn(0) = True Then   'towards
      MotionDirection = 0
   ElseIf optDirn(1) = True Then   'away
      MotionDirection = 1
   ElseIf optDirn(2) = True Then   'either towards or away
      MotionDirection = 2
   End If

   MotionFilterConstant = CSng(txtConstant.Text)
   MotionThreshold = CInt(txtThreshold.Text)


   'send command
   reply = Mailbox_send(WVFRM_MOTION_PARAMS)

   'transfer data
   MBoxPrintf (MotionDirection)
   MBoxPrintf (txtConstant.Text)
   MBoxPrintf (MotionThreshold)

   Exit Sub
Error0:
      msg = "Runtime error: " & Str(Err.Number)
      Title = "Fatal Error"
      MsgBox msg, 0, Title

End Sub

Private Sub cmdClose_Click()
   Me.Hide
End Sub

Private Sub Form_Activate()
'initialize form
   If SCANNING = True Then
      lblWarning.Visible = True
      cmdApply.Enabled = False
   Else
      cmdApply.Enabled = True
      lblWarning.Visible = False
   End If

   cmdClose.Enabled = True

   If MotionDirection = 0 Then
      optDirn(0).value = True
   ElseIf MotionDirection = 1 Then
      optDirn(1).value = True
   ElseIf MotionDirection = 2 Then
      optDirn(2).value = True
   End If

   txtThreshold = MotionThreshold
   txtConstant = MotionFilterConstant

End Sub
```

frmDisplay.frm

```
Private Sub cmdApply_Click()
   Dim midvalue As Integer

   X_Galvo_Scale_Factor = txtXScale.Text
   Y_Galvo_Scale_Factor = txtYScale.Text
   midvalue = (sldZoom.Max - sldZoom.Min) / 2
   Zoom = 2 ^ (sldZoom.value - midvalue)
End Sub

Private Sub cmdClose_Click()
   Me.Hide
End Sub

Private Sub Form_Activate()
   Dim midvalue As Integer

   txtXScale.Text = X_Galvo_Scale_Factor
   txtYScale.Text = Y_Galvo_Scale_Factor
   midvalue = (sldZoom.Max - sldZoom.Min) / 2
   sldZoom.value = (Log(Zoom) / Log(2)) + 5

End Sub
```

MailBox.bas

```
Attribute VB_Name = "Module1"
'Global definitions

'Target commands/responses must be identical to those recognized by the target

'Commands to Target.
Public Const WVFRM_SHOW_PINS As Integer = 1
Public Const WVFRM_SHOW_PARAMS As Integer = 2
Public Const WVFRM_DEFINE As Integer = 3
Public Const WVFRM_EDIT As Integer = 4
Public Const WVFRM_SELECT As Integer = 5
Public Const WVFRM_TEST_COMM As Integer = 6
Public Const WVFRM_GO As Integer = 7
Public Const WVFRM_STOP As Integer = 8
Public Const WVFRM_QUIT As Integer = 9
Public Const LAUNCH_TARGET As Integer = &HA
Public Const WVFRM_MOTION_PARAMS = &HB

'waveform editing commands
Public Const WVFRM_EDIT_AMPLITUDE As Integer = 1
Public Const WVFRM_EDIT_OFFSET As Integer = 2
Public Const WVFRM_EDIT_PHASE As Integer = 3
Public Const WVFRM_ESCAPE_CMD As Integer = &HF

'Responses from target
Public Const CMD_OK As Integer = 0
Public Const CMD_REJECTED As Integer = 1
Public Const CMD_UNKNOWN As Integer = 2

'Mailbox message types from target (multiplexed)
Public Const STRING_TYPE As Integer = 0
Public Const GRAPHIC_POINT_TYPE As Integer = 1
Public Const GRAPHIC_LINE_TYPE As Integer = 2
Public Const GRAPHIC_CIRCLE_TYPE As Integer = 3

'Message size from target
Public Const MAX_CHANGED_VOXELS_PER_LINE = 4096

Public Sub cmd_ack(ack As Long)

    If ack = CMD_OK Then
        frmMain.lblCmd.Caption = "Ok"
    ElseIf ack = CMD_REJECTED Then
        frmMain.lblCmd.Caption = "Wrong command"
    Else
        frmMain.lblCmd.Caption = "Error: " & ack
    End If

End Sub

Public Sub MBoxPrintf(OutVal As String)
    Dim length, i As Integer
    Dim OutChar As String

    'send length, but first wait for mailbox to become empty
    While frmMain.dsp.MailOutEmpty(0) = False
        ' wait till empty
    Wend

    length = Len(OutVal)
    frmMain.dsp.Mailbox(0) = length

    For i = 1 To length
        OutChar = Mid(OutVal, i, 1) 'dump string char by char
```

MailBox.bas continued

```
        While frmMain.dsp.MailOutEmpty(0) = False  'busywait if necessary
        Wend
        frmMain.dsp.Mailbox(0) = Asc(OutChar) 'coerce to int
    Next i

End Sub

Public Function Mailbox_send(out As Integer) As Long
    'this function busywaits for an empty mailbox and then
    'send the data.
    'it then busywaits for an acknowledgement

    While frmMain.dsp.MailOutEmpty(0) = False
        'wait till mailbox is empty
    Wend
    'send data
    frmMain.dsp.Mailbox(0) = out

    'wait till mailbox is full
    While frmMain.dsp.MailInFull(0) = False
    Wend
    'read data
    Mailbox_send = frmMain.dsp.Mailbox(0)

End Function

Public Function Mailbox_send_no_ack(out As Integer) As Long
    'this function busywaits for an empty mailbox and then
    'sends the data. It does NOT busywait for an acknowledgement
    'but returns a fake 'reply'

    While frmMain.dsp.MailOutEmpty(0) = False
        'wait while not empty
    Wend
    'send data
    frmMain.dsp.Mailbox(0) = out

    'return a fake reply
    Mailbox_send_no_ack = CMD_OK

End Function
```

Waveform.bas

```
Attribute VB_Name = "Module2"
'Waveform types:
Public Const COS As Integer = 0
Public Const SAW As Integer = 1
Public Const UNDEFINED As Integer = 2

Public Const X_AXIS As Integer = 0
Public Const Y_AXIS As Integer = 1

'Public Const MAX_AMP As Integer = 4
Public Const MAX_AMP As Integer = 16
Public Const MIN_AMP As Integer = 0
Public Const MAX_PHASE As Integer = 180
Public Const MIN_PHASE As Integer = -180
Public Const MAX_OFFSET As Integer = 5
Public Const MIN_OFFSET As Integer = -5

'Interface details
'Scrollbar limits
Public Const HSB_MAX_AMP As Integer = 1024
Public Const HSB_MIN_AMP As Integer = 0
Public Const HSB_MAX_PHASE  As Integer = 1024
Public Const HSB_MIN_PHASE  As Integer = 0
Public Const HSB_MAX_OFFSET  As Integer = 1024
Public Const HSB_MIN_OFFSET  As Integer = 0

'Other
Public Const two_pi As Single = 6.283185


'  Waveform specific definitions

'global variables
Public Const NumWaveforms As Integer = 8

'currently selected waveform:
Public WaveNum As Integer

' Current phase offsets for display
' These phase adjustments are used to align the display
' with the actual scanning waveforms
Public Display_Phase_Correction_X As Single
Public Display_Phase_Correction_Y As Single


' Motion detection parameters
Public MotionDirection As Integer
Public MotionThreshold As Integer
Public MotionFilterConstant As Single

' Motion detection constants
Public Const MOTION_DIR_POS As Integer = 0
Public Const MOTION_DIR_NEG As Integer = 1
Public Const MOTION_DIR_BOTH As Integer = 2
Public Const MOTION_DIFF_THRESHOLD As Integer = 3 ' initialization
Public Const MOTION_K_FACTOR As Single = 0.2   'initialization

'Waveform data structures
Public Type WaveInfo
    WaveType As Integer
    WaveAmp As Single
    WaveOffset As Single
    WavePhase As Single
    WavePhaseCorr As Single
```

Waveform.bas continued

```
    WaveVoxPerLine As Integer
    WaveCycles As Integer
    WaveRasterInc As Integer  '= 0 for first axis to be incremented (fast axis)
End Type

Public Type WaveDefine
    Wave_X As WaveInfo
    Wave_Y As WaveInfo
End Type

Public WaveRegistry(NumWaveforms) As WaveDefine

Public X_LUT(4096) As Single
Public Y_LUT(4096) As Single

'Galvo ADC constants
Public Const ADC_GAIN_CONSTANT As Single = 0.000305175   'volts/bit
Public X_Galvo_Scale_Factor As Integer
Public Y_Galvo_Scale_Factor As Integer
Public Zoom As Single




' Procedures

' this procedure initializes ALL waveforms with a common definition
Public Sub Init_Waveforms()
    Dim i As Integer

    For i = 0 To NumWaveforms - 1
        WaveRegistry(i).Wave_X.WaveType = COS
        WaveRegistry(i).Wave_Y.WaveType = COS
        WaveRegistry(i).Wave_X.WaveAmp = 10
        WaveRegistry(i).Wave_Y.WaveAmp = 2
        WaveRegistry(i).Wave_X.WaveOffset = 0
        WaveRegistry(i).Wave_Y.WaveOffset = 0
        WaveRegistry(i).Wave_X.WavePhase = 0
        WaveRegistry(i).Wave_Y.WavePhase = 4.5
        WaveRegistry(i).Wave_X.WavePhaseCorr = 0
        WaveRegistry(i).Wave_Y.WavePhaseCorr = 0
        WaveRegistry(i).Wave_X.WaveVoxPerLine = 2048
        WaveRegistry(i).Wave_Y.WaveVoxPerLine = 2048
        WaveRegistry(i).Wave_X.WaveCycles = 6
        WaveRegistry(i).Wave_Y.WaveCycles = 5
        WaveRegistry(i).Wave_X.WaveRasterInc = 0 'fast axis
        WaveRegistry(i).Wave_Y.WaveRasterInc = 1 'slow axis
    Next i

End Sub

Public Sub UpdateWaveDisplay()
' Updates display on waveform select/edit form

    If WaveRegistry(WaveNum).Wave_X.WaveType = COS Then
        frmSelect.txtType(0).Text = "COS"
    ElseIf WaveRegistry(WaveNum).Wave_X.WaveType = SAW Then
        frmSelect.txtType(0).Text = "SAW"
    Else
        frmSelect.txtType(0).Text = "?"
    End If

    If WaveRegistry(WaveNum).Wave_Y.WaveType = COS Then
```

Waveform.bas continued

```
        frmSelect.txtType(1).Text = "COS"
    ElseIf WaveRegistry(WaveNum).Wave_Y.WaveType = SAW Then
        frmSelect.txtType(1).Text = "SAW"
    Else
        frmSelect.txtType(1).Text = "?"
    End If

    frmSelect.txtAmp(0) = WaveRegistry(WaveNum).Wave_X.WaveAmp
    frmSelect.txtAmp(1) = WaveRegistry(WaveNum).Wave_Y.WaveAmp

    frmSelect.txtPhase(0) = WaveRegistry(WaveNum).Wave_X.WavePhase
    frmSelect.txtPhase(1) = WaveRegistry(WaveNum).Wave_Y.WavePhase

    frmSelect.txtPhCorr(0) = WaveRegistry(WaveNum).Wave_X.WavePhaseCorr
    frmSelect.txtPhCorr(1) = WaveRegistry(WaveNum).Wave_Y.WavePhaseCorr

    frmSelect.txtOffset(0) = WaveRegistry(WaveNum).Wave_X.WaveOffset
    frmSelect.txtOffset(1) = WaveRegistry(WaveNum).Wave_Y.WaveOffset

    frmSelect.txtPoints(0) = WaveRegistry(WaveNum).Wave_X.WaveVoxPerLine
    frmSelect.txtPoints(1) = WaveRegistry(WaveNum).Wave_Y.WaveVoxPerLine

    frmSelect.txtCycles(0) = WaveRegistry(WaveNum).Wave_X.WaveCycles
    frmSelect.txtCycles(1) = WaveRegistry(WaveNum).Wave_Y.WaveCycles

    If WaveRegistry(WaveNum).Wave_X.WaveRasterInc = 0 Then
        frmSelect.txtFast(0).Text = "Yes"
        frmSelect.txtFast(1).Text = "No"
    Else
        frmSelect.txtFast(0).Text = "No"
        frmSelect.txtFast(1).Text = "Yes"
    End If

End Sub

Public Sub UpdateWaveRegistry()

    If frmSelect.txtType(0).Text = "COS" Then
        WaveRegistry(WaveNum).Wave_X.WaveType = COS
    ElseIf frmSelect.txtType(0).Text = "SAW" Then
        WaveRegistry(WaveNum).Wave_X.WaveType = SAW
    Else
        WaveRegistry(WaveNum).Wave_X.WaveType = UNDEFINED
    End If

    If frmSelect.txtType(1).Text = "COS" Then
        WaveRegistry(WaveNum).Wave_Y.WaveType = COS
    ElseIf frmSelect.txtType(1).Text = "SAW" Then
        WaveRegistry(WaveNum).Wave_Y.WaveType = SAW
    Else
        WaveRegistry(WaveNum).Wave_Y.WaveType = UNDEFINED
    End If

    WaveRegistry(WaveNum).Wave_X.WaveAmp = Abs(CSng(frmSelect.txtAmp(0).Text))
    WaveRegistry(WaveNum).Wave_Y.WaveAmp = Abs(CSng(frmSelect.txtAmp(1).Text))

    WaveRegistry(WaveNum).Wave_X.WavePhase = CSng(frmSelect.txtPhase(0).Text)
    WaveRegistry(WaveNum).Wave_Y.WavePhase = CSng(frmSelect.txtPhase(1).Text)

    WaveRegistry(WaveNum).Wave_X.WavePhaseCorr = CSng(frmSelect.txtPhCorr(0).Text)
    WaveRegistry(WaveNum).Wave_Y.WavePhaseCorr = CSng(frmSelect.txtPhCorr(1).Text)

    WaveRegistry(WaveNum).Wave_X.WaveOffset = CSng(frmSelect.txtOffset(0).Text)
    WaveRegistry(WaveNum).Wave_Y.WaveOffset = CSng(frmSelect.txtOffset(1).Text)
```

Waveform.bas continued

```
    WaveRegistry(WaveNum).Wave_X.WaveVoxPerLine = CSng(frmSelect.txtPoints(0).Text)
    WaveRegistry(WaveNum).Wave_Y.WaveVoxPerLine = CSng(frmSelect.txtPoints(1).Text)

    WaveRegistry(WaveNum).Wave_X.WaveCycles = CSng(frmSelect.txtCycles(0).Text)
    WaveRegistry(WaveNum).Wave_Y.WaveCycles = CSng(frmSelect.txtCycles(1).Text)

    If frmSelect.txtFast(0).Text = "Yes" Then
        WaveRegistry(WaveNum).Wave_X.WaveRasterInc = 0
        WaveRegistry(WaveNum).Wave_Y.WaveRasterInc = 1
    Else
        WaveRegistry(WaveNum).Wave_X.WaveRasterInc = 1
        WaveRegistry(WaveNum).Wave_Y.WaveRasterInc = 0
    End If

End Sub

Public Sub DefineAllWaveformsCmd()
    Dim Index As Integer

    For Index = 0 To NumWaveforms - 1
        DefineWaveformCmd (Index)
    Next Index

End Sub

Public Sub DefineWaveformCmd(Index As Integer)

    'send command to update target
    frmMain.dsp.Mailbox(0) = WVFRM_DEFINE
    reply = frmMain.dsp.Mailbox(0)

    'check for command rejection
    If reply <> CMD_OK Then
        frmMain.lblFailed.Visible = True
        Exit Sub
    End If

    MBoxPrintf (Str(Index)) 'Wave index

    'X axis:
    MBoxPrintf (Str(WaveRegistry(Index).Wave_X.WaveType)) 'X wavetype
    If WaveRegistry(Index).Wave_X.WaveType = SAW Then
        If WaveRegistry(Index).Wave_X.WaveRasterInc = 0 Then
            MBoxPrintf ("0") '0 if X is fast axis
        Else
            MBoxPrintf ("1") '1 if Y is fast axis
        End If
    End If

    MBoxPrintf (Str(WaveRegistry(Index).Wave_X.WaveAmp))          'X Amplitude
    MBoxPrintf (Str(WaveRegistry(Index).Wave_X.WaveOffset))       'X Offset
    MBoxPrintf (Str(WaveRegistry(Index).Wave_X.WavePhase))        'X Phase
    MBoxPrintf (Str(WaveRegistry(Index).Wave_X.WavePhaseCorr))    'X Phase Correction
    MBoxPrintf (Str(WaveRegistry(Index).Wave_X.WaveVoxPerLine))   'X points
    MBoxPrintf (Str(WaveRegistry(Index).Wave_X.WaveCycles))       'X cycles

    'Y Axis:
    MBoxPrintf (Str(WaveRegistry(Index).Wave_Y.WaveType))         'Y wavetype
    MBoxPrintf (Str(WaveRegistry(Index).Wave_Y.WaveAmp))          'Y Amplitude
    MBoxPrintf (Str(WaveRegistry(Index).Wave_Y.WaveOffset))       'Y Offset
    MBoxPrintf (Str(WaveRegistry(Index).Wave_Y.WavePhase))        'Y Phase
    MBoxPrintf (Str(WaveRegistry(Index).Wave_Y.WavePhaseCorr))    'Y Phase Correction
    MBoxPrintf (Str(WaveRegistry(Index).Wave_Y.WaveVoxPerLine))   'Y points
    MBoxPrintf (Str(WaveRegistry(Index).Wave_Y.WaveCycles))       'Y cycles
```

Waveform.bas continued

```
End Sub

Public Sub BuildLUT_X()

'Builds Lookup table for x-axis COS waveform.

   Dim cycles As Integer
   Dim line_size As Single

   cycles = WaveRegistry(WaveNum).Wave_X.WaveCycles
   line_size = WaveRegistry(WaveNum).Wave_X.WaveVoxPerLine
   For i = 0 To line_size - 1
      X_LUT(i) = Sin((two_pi * i * cycles / line_size) + 1.5708)
   Next i
End Sub

Public Sub BuildLUT_Y()

'Builds Lookup table for y-axis COS waveform.

   Dim cycles As Integer
   Dim line_size As Single

   cycles = WaveRegistry(WaveNum).Wave_Y.WaveCycles
   line_size = WaveRegistry(WaveNum).Wave_Y.WaveVoxPerLine
   For i = 0 To line_size - 1
      Y_LUT(i) = Sin((two_pi * i * cycles / line_size) + 1.5708)
   Next i
End Sub

Public Sub Get_Display_Phase()

' Sets and returns (x and y) phase of displayed waveform for proper alignment
' with actual scanning waveforms.
' The actual phase corrections were determined experimentally.

   Dim x_cycles As Integer
   Dim y_cycles As Integer
   Dim points As Integer

   x_cycles = WaveRegistry(WaveNum).Wave_X.WaveCycles
   y_cycles = WaveRegistry(WaveNum).Wave_Y.WaveCycles
   points = WaveRegistry(WaveNum).Wave_Y.WaveVoxPerLine

   'defaults:
   Display_Phase_Correction_X = 0
   Display_Phase_Correction_Y = 0

   If WaveRegistry(WaveNum).Wave_X.WaveType <> COS Then
      Display_Phase_Correction_X = 0
   End If

   If WaveRegistry(WaveNum).Wave_Y.WaveType <> COS Then
      Display_Phase_Correction_Y = 0
   End If

   '2048 point lines:
   If WaveRegistry(WaveNum).Wave_Y.WaveVoxPerLine = 2048 Then

      Select Case WaveRegistry(WaveNum).Wave_X.WaveCycles
         Case 0
            Display_Phase_Correction_X = 0
```

Waveform.bas continued

```
        Case 1
           Display_Phase_Correction_X = 121
        Case 2
           Display_Phase_Correction_X = 78
        Case 3
           Display_Phase_Correction_X = 33
        Case 4
           Display_Phase_Correction_X = 350
        Case 5
           Display_Phase_Correction_X = 315
        Case 6
           Display_Phase_Correction_X = 277
     End Select


     Select Case WaveRegistry(WaveNum).Wave_Y.WaveCycles
        Case 0
           Display_Phase_Correction_Y = 0
        Case 1
           Display_Phase_Correction_Y = 325
        Case 2
           Display_Phase_Correction_Y = 290
        Case 3
           Display_Phase_Correction_Y = 250
        Case 4
           Display_Phase_Correction_Y = 215
        Case 5
           Display_Phase_Correction_Y = 177
        Case 6
           Display_Phase_Correction_Y = 140
     End Select
   End If

End Sub
```

App.bas

```
'General definitions

Public SCANNING As Boolean
```

Appendix D. Installing Precise/MQX on a Q67 or M62 DSP Platform

This appendix describes the installation and development of an initial board support package (BSP) of MQX2.40 on either a Quatro67 (Q67) card or an M62 card. Differences in the two installations if any, are clearly indicated. In addition to the basic port described here, the full BSP also required the design and implementation of the interprocessor communications (IPC) mechanism. The description of the IPC design is beyond the scope of this report.

The release of MQX 2.40 that was originally delivered contained the 3206701L PSP (i.e. for the C6701, little endian mode) and the TIEVM67L BSP (the BSP for the TI EVM card for the C6701). These demonstration ports had to be modified for operation on the Q67/M62 hardware.

Some additional files, apart from those contained on the Precise/MQX distribution CD, are required for the installation. The folders within the Precise directory path containing these files are tievm60 and tidsk211.

The Q67 and M62 boards were designed for big endian operation. This version of MQX 2.40 was developed and tested by Precise in little endian mode only. The PSP, BSP and test programs can all be built in big endian mode as indicated below. Behaviour in big endian mode appears to be normal.

Installation and testing of MQX2.40

The installation of MQX involves the following steps:

- Install all board packages and toolsets and then test the hardware (following the manufacturers' recommendations) before commencing MQX installation.
- install the MQX processor support package (PSP).
- install the MQX board support package (BSP).
- build the PSP library (..\precise\mqx2.40\lib\3206x01b.ti\mqx.lib).
- verify DOS environment variables.
- modify the BSP.
- build the BSP library (..\precise\mqx2.40\lib\iix6xb.ti\iix6x.lib).
- test the installation using the MQX provided build scripts.
- initialize the Code Composer Studio (CCS) environment.
- Adding I/O support for Innovative's PCI-based terminal emulator.
- CodeWright Upgrade (Vers. 6.0b to 6.0c).
- Synchronizing Code Composer Studio with CodeWright (optional).
- Using the Code Composer Studio Development Environment (optional).

A. Install PSP and BSP

1. The following default directory structure is assumed:
   C:\precise\mqx2.40\source\bsp

2. Create the following directories:
   ..\source\bsp\tidsk211
   ..\source\bsp\tievm60

   If using the Q67, create
   ..\source\bsp\iiq67

   or for the M62 create
   ..\source\bsp\iim62.

   Note: In the following, ***iix6x*** refers to either of the above directories.

3. Install PSP and BSP as described in the MQX Release Notes.

B. Update the PSP

To correct a possible problem with the C6x timer interrupt, the following file must be adjusted as indicated: …\source\psp\32060\dispatch.ti

```
> At line 425:
>
>       LDW     *+B15[PSP_REG_B12],B12
>     || LDW     *+A15[PSP_REG_A12],A12
>     || CLR     B4,0,1,B4                ; clear PGIE and GIE bit
>
>       LDW     *+B15[PSP_REG_B13],B13
>     || LDW     *+A15[PSP_REG_A13],A13
>     || [!B1]   OR 2,B4,B4                ; set PGIE bit ie ...
>                              ... interrupts will be enabled
>
>       LDW     *+B15[PSP_REG_AMR],B9        ; get AMR
>       STH     A5,*+B8(KD_ACTIVE_SR)
>
> Change this to:
>
>       LDW     *+B15[PSP_REG_B12],B12
>     || LDW     *+A15[PSP_REG_A12],A12
>     || CLR     B4,0,0,B4                ; ??? clear only GIE bit
>
>       LDW     *+B15[PSP_REG_B13],B13
>     || LDW     *+A15[PSP_REG_A13],A13
>     || NOP                          ; ??? leave PGIE as is
>       LDW     *+B15[PSP_REG_AMR],B9        ; get AMR
>       STH     A5,*+B8(KD_ACTIVE_SR)
>
```

To correct a dispatch problem, at line 244 change:

```
        STW     B11,*+B15[PSP_REG_B11]
```

to:

```
        STW     B6,*+B15[PSP_REG_CSR]
```

And at line 269 change:

```
        STW     B6,*+B15[PSP_REG_CSR]   ; store CSR
```

to:

```
        STW     B11,*+B15[PSP_REG_B11]   ; store CSR
```

[2001-05-24]
make the following corrections (C6x revisions):

```
__sched_execute_scheduler_internal:
        ADDK    -PSP_STACK_FRAME_SIZE,B15
        STW     B14,*+B15[PSP_REG_B14]   ; save data page pointer
        GET_KERNEL_DATA B8              ; Get address of kernel data (3 instr)
        MVC     CSR,B6                  ; get status register
        STW     A15,*+B15[PSP_REG_A15]   ; save rest of scratch registers
;changed 2001-05-24 telecon Mati Sauks (#179)
;   || CLR     B6,0,1,B6               ; clear GIE and PGIE bits
    || CLR     B6,0,1,B7               ; clear GIE and PGIE bits

        STW     B13,*+B15[PSP_REG_B13]
    || MV      B15,A15
        STW     B12,*+B15[PSP_REG_B12]
;changed 2001-05-24 telecon Mati Sauks (#179)
;   || MVC     B6,CSR                  ; disable interrupts
    || MVC     B7,CSR                  ; disable interrupts
    || STW     A14,*+A15[PSP_REG_A14]
```

[2001-05-25]
THIS DOESN'T WORK – for reference only.
to turn off interrupt nesting, make the following changes

```
;turn off nesting 2001-05-25 telecon, M.S.
;       MVC     B9,IER
        NOP
    || ADD      A1,A8,A7
        MVC     B7,CSR
```

And:

```
; Start SPR P154-0100-01
;****      NOP     2

;Turn off nesting 2001-05-25 telecon M.S.
;       MVC     A6,IER
        NOP
; End SPR P154-0100-01
```

New files and other changes are required to correct a problem with nested interrupts. The interrupt enable register (IER) is not adjusted during nesting. The following changes correct this.

In ...\source\psp\32060\32060.h

add field to `typedef struct psp_interrupt_frame_struct`

```
uint_32    IER;
```

add function prototypes:

```
extern void _psp_clr_ier(uint_32);
extern void _psp_set_ier(uint_32);
```

create new files clr_ier.c, set_ier.c (see source code backup for details).

In …\source\psp\32060\comp.bat

Add build command line:

```
call %mqx_cmd% clr_ier set_ier
```

C. Build the PSP

NOTE:

With TI C compiler Vers. 4.0, it may be necessary to execute this procedure twice since environment variables are adjusted during the first pass.

Execute `..\precise\mqx2.40\tools.bat` to define the DOS environment variables.

Build the PSP (use big endian mode for M62/Q67 hardware):

In c:\Precise\MQX2.40\build

Enter: `go 3206201 ti big` (M62)

`go 3206701 ti big` (Q67)

D. Verify DOS environment variables

Windows98 appears to limit the size of the environment space available. Less trouble has been experienced with Win95.

In config.sys

- Change command line to read `c:\command.com /p /e:8192`

Check autoexec.bat to ensure that at least the minimum necessary pathnames are included. The IDE is very fragile and will not work if the environment variables are incorrect. See the lists for the M62 and Q67 which are attached at the end of this Appendix.

E. Modify the BSP

Copy files from \tievm60 to \iix6x. Copy all ti_*.h files from \tidsk211 to \iix6x.

For the M62, edit all files in \iim62, replace all references to “tievm60” with “iim62”.

For the Q67, edit all files in \iiq67, replace all references to “tievm60” with “iiq67”.

Change tievm60.h to iim62.h or iiq67.h.

Edit comp.bat.

- Add a copy of ti_timer.h and ti_intr.h to the \build directory.
- Comment out calls to ‘enet’, ‘enet.ini’
- Comment out copy to enet.h, esonic.h
- Add copy to ti_regs.h, ti_timer.h, ti_intr.h, bsp_prv.h init6x*.gel, restart*.gel

Note:
For compiling with either CCS or CW, mqx_init should be excluded from iix6x.lib. This can be done by commenting out the appropriate line in comp.bat (“..mqx_init”). This is note does not apply when the Precise build tools are used directly.

Edit bsp_prv.h. Delete reference to evm_init().

Edit bsp.h

- add #include statements for ti_timer.h and ti_intr.h,
- comment out include statements for \dsp\include\timer.h and \dsp\include\intr.h

Edit iiq67.h / iim62.h:

- Add the appropriate line:

```
#define MQX_CPU 3206701
#define MQX_CPU 3206201
```

- Comment out all defines specific to the TI EVM board:
  i.e. everything from the first
  ```
  #ifdef USE_MAP_0
  ```
  to the final
  ```
  BSP_SDCNTL_REG
  ```
  and
  ```
  #endif /* USE_MAP_0*/.
  ```

- Change:
  from `#define BSP_TIEVM60`
  to `#define BSP_IIQ67` or
  `#define BSP_IIM62`

- Update the memory map:

```
#define BSP_INTERNAL_BASE              0x00000000
#define BSP_PERIPHERALS_BASE           0x01800000
#define BSP_SDRAM_BASE                 0x02000000
#define BSP_SDRAM2_BASE                0x03000000
#define BSP_INTERNAL_DATA_RAM_BASE     0x80000000
```

- Update ISTP:

```
#define BSP_ISTP_BASE_ADDRESS          0x02000000
```

- Update Default MQX Initialization Definitions. Only MAP_1 is used:

```
#define MQX_DEFAULT_START_OF_KERNEL_MEMORY    0x02FC0000
#define MQX_DEFAULT_END_OF_KERNEL_MEMORY      0x02FFFFFF
```

- Delete Ethernet definitions

- Change timer0 interrupt level. This is required because the Innovative libraries configure the FIFOLink interrupts to C6x external interrupts (levels 4-7):
  From:

```
#define BSP_TIMER_INTERRUPT_VECTOR PSP_EXCPT_INT4
```

  To:

```
#define BSP_TIMER_INTERRUPT_VECTOR PSP_EXCPT_INT14
```

- Change system clock definition:
  From:

```
#define BSP_SYSTEM_CLOCK   _bsp_get_cpu_speed()
```

  To:

```
#define BSP_SYSTEM_CLOCK   (160000000UL)
```

Edit ti.c.

- Remove reference to evm_init() and intr.h.
- **Do NOT perform the following step if TI C compiler Vers. 4.0 is being used.**
  Copy function intr_map() from ..\bsp\tidsk211\ti.c into \ti.c

Edit init_bsp.c

- In _bsp_enable_card() check for call to proper functions and arguments:
  For Q67:

```
_mqx_set_cpu_type(PSP_CPU_TYPE_TMS320C6701);
_tms3206701_initialize_support();
```

  For M62:

```
_mqx_set_cpu_type(PSP_CPU_TYPE_TMS320C6201);
_tms3206201_initialize_support();
```

- CCS must be restarted when MQX applications are shutdown. An attempt to solve this problem is made as follows:

  Add `include c:\q6x\include\target\periph.h` (Q67) or
  `include c:\m6x\include\target\periph.h` (M62).

  In _bsp_enable_card() add:

  [2001-07-04] Removed all stdio functions in the Q6x bsp. This allows proper operation using II's DSPComponent ActiveX component.
  Comment out all related functions:
  For Q67:

```
//  If( cpu_number() == 0 ) stdio_reset();
```

For M62:

```
Stdio_reset();
```

In bsp_exit_handler() add:

```
_psp_icache_disable();
```

This allows MQX applications to restart correctly under Code Composer Studio.

For Q67:

```
If( cpu_number() == 0 )
{
[2001-07-04] comment out Q6x stdio functions as stated above.
//  getchar();
//  stdio_terminate();
    exit(0);
}
```

for M62:

```
stdio_terminate();
```

- To support FIFOLink interrupts, EXT_INT4-7 must be configured for inverting polarity. Add the following lines:

```
INTR_EXT_POLARITY(0,1); //EXT_INT4
INTR_EXT_POLARITY(2,1); //EXT_INT6
INTR_EXT_POLARITY(3,1); //EXT_INT7
```

For Q6x:

```
INTR_EXT_POLARITY(1,1); //EXT_INT5
```

For M6x

```
INTR_EXT_POLARITY(1,0); //EXT_INT5
```

As delivered, the **clock timer** does not function. The following changes correct this problem.

- In _bsp_get_cpu_speed() change to read:

```
uint_32    speed;

speed = BSP_SYSTEM_CLOCK;
return speed;
```

- In bsp_timer_isr() add the following to clear the timer interrupt (***the cpu will hang if this change is not implemented!***):

```
INTR_CLR_FLAG(BSP_TIMER_INTERRUPT_VECTOR);
```

Edit link.ti

- change memory map:

  Note: do not use "//" for comments in this file.

```
/* Map 1 */
IPM      o = 00000000h  l = 00010000h      /* cache */
IDM      o = 80000000h  l = 00010000h      /* internal data */
INT      o = 02000000h  l = 00000200h      /* int vecs */
SDRAM0   o = 02000200h  l = 00fbfe00h
SDRAM1   o = 02fc0000h  l = 00040000       /* kernel data */
```

  For M62:

```
ASRAM    o = 01600000h  l = 00080000h      /* PCI interface */
```

  For Q67:

```
SBSRAM   o=03000000h    l = 00020000h      /* 128kW */
ASRAM    o = 01700000h  l = 00080000h      /* PCI interface p0 */
```

  .text, .cinit, .far, .ipmtext, .vec are all assigned to SDRAM0
  .vectors is assigned to INT
  all others are assigned to IDM.

- add appropriate Innovative Int. libraries (m6x or q6x)

```
c:\m6x\lib\target\stdio.lib
c:\m6x\lib\target\periph.lib
```

Edit \bsp\source\bspsetup.bat

```
mqx_proc=3206701 (or 3206201)
```

Edit bsp_lib.bat

- comment out all lines(!)

F. Build BSP

Depending on the autoexec.bat file contents (see above), it may be necessary to execute these commands in a DOS window before building the BSP:

```
cd c:\precise\mqx2.40
tools
```

Now build the BSP (big endian mode)

```
cd c:\Precise\MQX2.40\build
go iim62 ti big (for the M62),
go iiq67 ti big (for the Q67)
```

G. Test the Installation

Build the demo "hello":

```
cd c:\Precise\MQX2.40\examples\hello
go iim62 ti big
```

Results will appear in subfolder
..\Precise\MQX2.40\example\hello\iim62b.ti

Generating the output file hello.out indicates that the 'build' mechanism is working.

H. Setup Code Composer Studio Environment

Before running the demo, a GEL script file has to be prepared for Code Composer Studio. This script enables SDRAM for downloading.

For the M62, Edit
c:\precise\mqx2.40\source\bsp\iim62\init6x.gel

Add the following:

```
GEL_MemoryFill(0x1800000,0,0x1,0x3069);
GEL_MemoryFill(0x1800004,0,0x1,0x73E70F22);
GEL_MemoryFill(0x1800008,0,0x1,0x11010410);
GEL_MemoryFill(0x1800010,0,0x1,0x30);
GEL_MemoryFill(0x1800014,0,0x1,0x40);
GEL_MemoryFill(0x1800018,0,0x1,0x07117000);
GEL_MemoryFill(0x180001c,0,0x1,0x618);
GEL_MemoryFill(0x19c0008,0,0x1,0xF);
```

For the Q67, edit
c:\precise\mqx2.40\source\bsp\iiq67\init6x0.gel

For Processor 0 use the following:

```
/* GEL_MemoryFill(0x1800000,0,0x1,0x306A); */
GEL_MemoryFill(0x1800000,0,0x1,0x3069);

GEL_MemoryFill(0x1800004,0,0x1,0x73E70F22);

/* GEL_MemoryFill(0x1800008,0,0x1,0x30810420); */
GEL_MemoryFill(0x1800008,0,0x1,0x30800520);

GEL_MemoryFill(0x1800010,0,0x1,0x30);
GEL_MemoryFill(0x1800014,0,0x1,0x40);
GEL_MemoryFill(0x1800018,0,0x1,0x07117000);
GEL_MemoryFill(0x180001c,0,0x1,0x618);
GEL_MemoryFill(0x19c0008,0,0x1,0xF);
```

For processors 1-3, use the following:
c:\precise\mqx2.40\source\bsp\iiq67\init6x1.gel

```
/* GEL_MemoryFill(0x1800000,0,0x1,0x306A); */
```

```
GEL_MemoryFill(0x1800000,0,0x1,0x3069);

/* GEL_MemoryFill(0x1800004,0,0x1,0x30810420); */
GEL_MemoryFill(0x1800004,0,0x1,0x73E70F22);

/* GEL_MemoryFill(0x1800008,0,0x1,0x30810420); */
GEL_MemoryFill(0x1800008,0,0x1,0x30800520);

GEL_MemoryFill(0x1800010,0,0x1,0x30);
GEL_MemoryFill(0x1800014,0,0x1,0x40);
GEL_MemoryFill(0x1800018,0,0x1,0x07117000);
GEL_MemoryFill(0x180001c,0,0x1,0x618);
GEL_MemoryFill(0x19c0008,0,0x1,0xF);
```

Install the above gel files into CCS to enable external memory for download. See CCS->Help for details in “Auto-executing GEL functions”. For the Q6x, this procedure uses the “options->startup” menu from the parallel debug manager. For the M6x, use the “File | Load Gel” menu.

Load and run hello.out

For the M62, launch the following Innovative Integration applets (this procedure is especially important when the emulator and target reside on different host PCs).

- JTAG diagnostic
- boot.exe
- UniTerminal
- Launch CCS
- Load program hello.out
- Output should appear in the stdout window of CCS

For the Q67:

- qboot.exe
- UniTerminal
- Launch CCS
- From the parallel debug manager, File->load program, navigate and download hello.out to all 4 cpus.
- Click ‘Run’ on the parallel debug manager toolbar.
- A succesful build will cause each of the 4 processors to output “Hello World” to its CCS stdio window.

I. Adding I/O support for Innovative’s PCI-based terminal emulator.

The polled i/o driver used for the C6x package is derived from the xuart.c example (see c:\precise\mqx2.40\source\io\serial\polled\xuart.c). (Note that interrupt operation is not possible due to the design of the terminal emulator software.) A new version of the driver for the IIM62 was created (see ..\source\io\serial\polled\iim62io.c and ..\source\io\iim62io.h). Low level I/O is performed by calling functions provided by Innovative Int.

Copy ..\precise\mqx2.40\source\io\xuart.h to iix6xio.h (i.e. iiq67io or iim62io).

Copy ..\precise\mqx2.40\source\io\serial\polled\xuart.c to iix6xio.c.

Edit iix6xio.c

- Change all references to "xuart" to iiq67io or iim62io as appropriate.
- Edit each function contained in the file:

  - *iix6xio_serial_polled_init()*
    remove contents of body.

  - *iix6xio_serial_polled_deinit()*
    remove contents of body.

  - *iix6xio_serial_polled_getc()*
    ```
    return (uchar)kbd_key();
    ```

  - *iix6xio_serial_polled_putc()*
    ```
    emit(c);
    ```

  - *iix6xio_serial_polled_status()*
    ```
    if ( kbd_hit() ) return TRUE;
    return FALSE;
    ```

Edit iix6xio.h
- Change all references to xuart to iiq67io or iim62io.
- Remove all constant definitions
- Remove Datatype definitions except for *iiq67io_init_struct*
- Change #ifndef _xuart_h statements.

Edit iix6x.h
```
#define BSP_DEFAULT_IO_CHANNEL      "ttya:"
#define BSP_DEFAULT_IO_OPEN_MODE    (pointer)(IO_SERIAL_TRANSLATION | IO_SERIAL_ECHO)
```

Edit c:\precise\mqx2.40\source\bsp\iim62\init_bsp.c
Install serial device drivers for the console and for CCS (Q67 only).
- #include "periph.h" if not already included.
- In _bsp_enable_card() for Q67
  ```
  if( cpu_number() == 0 )
      _iiq67io_serial_polled_install( "ttya:, NULL, 1 );
  else
      _io_serial_polled_go_dsp_install( "ttya:");
  ```

- In _bsp_enable_card() for M62, remove reference to godsp, use:
  ```
  _iim62io_serial_polled_install( "ttya:, NULL, 1 );
  ```

Edit comp.bat

```
call ..\source\io\serial\polled\iix6xio
copy ..\source\io\iix6xio.h
```

for Q67, copy gel files too

```
copy ..\source\bsp\iix6x\init6x*.gel
```

Rebuild the BSP (see Section E. above).

Test the installation with the MQX version of hello.c. For the Q67, loading the test file into p0, output should appear on the terminal emulator screen. If the program is loaded into any of the other 3 processors, i/o will be handled by stdio windows in CCS.

J. CodeWright Upgrade (Vers. 6.0b to 6.0#)

Download appropriate upgrade(s) (x3260b_#.exe) from the Premia website: http://www.premia.com/support/codewright/patches/index.html. This is saved locally in c:\cw32\patches. CodeWright Vers.6.0b is installed originally. Current upgrade as of 2000-09-06 is Vers. 6.0e.

K. Using the CodeWright Development Environment (Optional)

Innovative Integration has developed an integrated development environment based on CodeWright. A utility is provided which automatically generates makefiles (*.mk) based on a "make include file" (*.mki). This mechanism is described in detail in the M62 and Q67 software manuals. Each project has its own *.mki file which is derived from generic.mki file found in the board root directory (..\m6x\generic.mki or ..\q62cc\generic.mki). This generic file should be edited for the commonly used parameters used for compiling and linking applications.

For the M62 MQX applications, edit generic.mki:

```
CC_ARGS = -g -q -x2 -o2 -me -ml2
LNK_ARGS = -c -stack 0x800 -heap 0x0800000 –x
LIBS = -l stdio.lib -l periph.lib -l dsp.lib -l rts6201e.lib -l iim62.lib -l mqx.lib
```

For the Q67 MQX applications, edit generic.mki:

```
CC_ARGS = -g -q -x2 -o2 -me -ml2 -mv6701
LNK_ARGS = -c -stack 0x800 -heap 0x0800000 -x
LIBS = -l stdio.lib -l periph.lib -l dsp.lib -l rts6701le.lib -l iiq67.lib -l mqx.lib
```

L. Synchronizing Code Composer Studio with CodeWright (Optional)

A plug-in utility exists for Code Composer Studio which allows CodeWright to be used as the default editor and which synchronizes access to source files.

Download tisscync.zip from
http://www.premia.com/support/codewright/addons/index.html. This is saved locally in c:\cw32\TiCCSync.

Unzip and follow the installation instructions in Ticcsync.txt.

M. Using the Code Composer Studio Development Environment (Optional)

CCS can be used as a development environment rather than CodeWright. The major advantage is that a GUI is provided to simplify the selection of compiler options for each project. The disadvantage is that there is no automatic makefile generation so the compiler options, library search paths and libraries must be manually specified when the project is first created. CodeWright's automatic generation of makefiles is far simpler to use. CCS does offer *Profile Based Compilation* (PBC) which can be used to tune the application for code size vs performance. DSP/BIOS may also be of interest especially for non-obtrusive performance analysis. For details on the use of CCS, see *The M62 Development Package Software Manual* from Innovative Integration, p. 74-81.

Development of MQX2.40 applications using CCS generates an error during linking: "multiply defined: MQX_init_struct". The source of the error is the file C:\mqx2.40\lib\iim62b.ti\mqx_init.obj which should be deleted. It is only required for compiling MQX test programs using Precise- supplied scripts. It can safely be removed from the BSP build batch file.

N. Installing Precise Solution

Precise Solution contains documentation and debug tools that should be installed. Install the CD-ROM and navigate to the ..\solution directory. Launch "Setup" to install the package.

O. Installing MQX Task Aware Debugging Plug-in for Code Composer Studio

See the "Precise Task Aware Debug User's Manual", page 33 for installation and use.

P. Required Patches after Reinstalling the Q67 Development Package

In ..\q62cc\include\target\ii_c6x.h
Redfine TRUE and FALSE definitions to avoid clashes with MQX:

```
# ifdef TRUE
# undef TRUE
# endif
# define TRUE -1
```

Create q62.h:

```
Define __Q62__
#include periph.h
```

this is required by misc.h to define the fifolink interrupt offsets.

Edit generic.mki:

```
CC_ARGS = -g -q -x2 -o2 -me -ml2 -mv6701
LNK_ARGS = -c -stack 0x800 -heap 0x0800000 –x
LIBS = -l stdio.lib -l periph.lib -l dsp.lib -l rts6701le.lib -l iiq67.lib -l mqx.lib
```

Q. Special Note Regarding Installation of Innovative Integration "Zuma" Toolsets

Zuma Toolsets for the M6x starting with Zuma 1.18 and for the Q6x starting with Vers. 1.07 must use TI C compiler vers. 4.0 which is shipped with Code Composer Studio vers. 1.2. My experience with these versions of the toolsets is that the processors fail to start correctly when MQX libraries are linked to an application. In all cases, processor initialization failed during the autoinitialization sequence when initialized variables are being setup. This failure occurs before application code is called.

The fix is to copy ..\m6x\periph\rts\boot.c from a previous version of the Zuma toolset (M6x Zuma Vers. 1.16 or Q6x Zuma Vers. 1.06) to the current directory and then to rebuild the Innovative libraries. This seems to correct the startup problem. Note that the same boot.c file is found in M6x and Q6x packages.

Building Innovative Libraries.
To rebuild the Innovative Libraries:

In Windows explorer, navigate to ..\m6x or ..\q6x as appropriate,
Double click "relib".

If the compiler reports errors, check to be sure that the necessary environment variables have been defined in autoexec.bat. It may be necessary to modify the "c_dir" environment variable to include any missing compiler search paths (e.g. ..\ti\c6000\bios\include…) etc.

R. Note Concerning Library Functions: fopen(), fclose(), fflush(), fwrite().

[2001-03-12]
Innovative uses fopen() to create or access a DOS file on the PC. Precise uses io_fopen() for an entirely different purpose. Unfortunately, Precise aliases their io_fopen() with fopen() so the II function is not available as is. The fix is simple:

Navigate to c:\q6x\stdio\fopen.c
Change the function name to ii_fopen().
Navigate c:\q6x\include\target\stdio.h
Change the prototype from fopen() to ii_fopen().

Repeat for fclose() and fflush().
Rebuild the libs as described above.

Bug fix for Innovative function fwrite().
[2001-03-15]
fwrite() should be able to write to PC files performing 4-byte, 3-byte, or 2-byte transfers. The existing 2-byte version fails on the C6x. It has been corrected as follows:

```
case 2:
{
    switch (sizeof(long))
    {
         // C3x, C4x
        case 1:
        {
            emit(*addr);
            emit(*addr >> 8);
         }
            break;

         case 8:
        {
            // C6x
            emit(*(addr + 3));
            emit(*(addr + 2));
            addr += 4;
         }
            break;
      }
            break;
  }
```

S. M62 Environment Variables (autoexec.bat) for Zuma Toolset 1.16

Proper selection of DOS evnironment variables is essential for correct operation of the code generation tools. Windows 98 appears to limit the number of variables that can be specified. The work-around was to eliminate non-essential entries. These are identified below by strikethrough text.

```
SET ii_board=          C:\M6x

SET d_dir=             C:\c6xtools\lib

SET d_src=             C:\M6x;
                       C:\M6x\stdio;
                       C:\M6x\periph\bus;
                       ~~C:\M6x\periph\rts;~~
                       C:\M6x\periph\digital;
                       C:\M6x\periph\misc;
                       C:\M6x\dsp;
                       C:\M6x\periph\analog

SET path=              %PATH%;
                       %c_dir%;
                       c:\ti\c6000\cgtools\bin;  /* ← vers. TI C compiler Vers.3.01
                       c:\ti\bin;
                        C:\M6x;
                       C:\M6x\lib\host

SET MQX_ROOT=          C:\PRECISE\MQX2.40

SET MQX_BAT_ROOT= C:\PRECISE\MQX2.40\BUILD

SET MQXC_ROOT=         C:\TI\C6000\CGTOOLS

SET c_dir_1=           C:\M6X;
                       C:\M6X\INCLUDE\TARGET;
                       C:\M6X\LIB\TARGET

SET c_dir=             %c_dir_1%;
                       ~~C:\TI\C6000\RTDX\INCLUDE~~;          ← required for II lib rebuild
                       ~~C:\TI\C6000\RTDX\LIB;~~
                       C:\TI\C6000\CGTOOLS\INCLUDE;
                       C:\TI\C6000\CGTOOLS\LIB;
                       ~~C:\TI\C6000\BIOS\INCLUDE;~~          ← required for II lib rebuild
                       ~~C:\TI\C6000\BIOS\LIB;~~
                       C:\precise\mqx2.40\lib\3206201b.ti;
                       C:\precise\mqx2.40\lib\iim62b.ti

SET a_dir=             %C_DIR%
```

T. Q67 Environment Variables (autoexec.bat) for Zuma Toolset 1.06

Environment variables for the Quatro67 using Zuma Toolset 1.06 now use TI C compiler vers. 3.01. The previous release used C compiler vers. 2.0.

```
SET ii_board=           C:\Q6x

SET d_dir=              c:\c6xtools\lib

SET d_src=              C:\Q6x;
                        C:\Q6x\stdio;
                        C:\Q6x\periph\bus;
                        C:\Q6x\periph\digital;
                        C:\Q6x\periph\misc;
                        C:\Q6x\dsp;
                        C:\Q6x\periph\fifo

REM The following builds a path for TI C compiler vers.3.01
SET path=               %path%;
                        c:\ti\c6000\cgtools\bin;
                        c:\ti\bin;
                         C:\Q6x;
                        C:\Q6x\lib\host

call c:\precise\mqx2.40\tools.bat

REM *** SETTINGS FOR TMS320C6x Code Generation Tools Release 3.01 ***
REM Uncomment for Quatro67 board:
SET c_dir=              C:\Q6X;
                        C:\Q6X\INCLUDE\TARGET;
                        C:\Q6X\LIB\TARGET;
                        C:\TI\C6000\CGTOOLS\INCLUDE;
                        C:\TI\C6000\CGTOOLS\LIB;
                        C:\PRECISE\MQX2.40\LIB\3206701b.ti;
                        C:\PRECISE\MQX2.40\LIB\iiq67b.ti

REM Uncomment for Quatro62 Board:
REM SET c_dir=          C:\Q6X;C:\Q6X\INCLUDE\TARGET;
                        C:\Q6X\LIB\TARGET;
                        C:\TI\C6000\CGTOOLS\INCLUDE;
                        C:\TI\C6000\CGTOOLS\LIB;
                        C:\PRECISE\MQX2.40\LIB\3206201b.ti;
                        C:\PRECISE\MQX2.40\LIB\iiq62b.ti

SET a_dir=%C_DIR%

REM for reference - original versions
REM SET c6x_c_dir=      C:\Q6X;
                        C:\Q6X\INCLUDE\TARGET;C:\Q6X\LIB\TARGET;
                        C:\TI\C6000\RTDX\INCLUDE;
                        C:\TI\C6000\RTDX\LIB;
                        C:\TI\C6000\CGTOOLS\INCLUDE;
                        C:\TI\C6000\CGTOOLS\LIB;
                        C:\TI\C6000\BIOS\INCLUDE;
                        C:\TI\C6000\BIOS\LIB

REM SET c6x_ a_dir=%C6X_C_DIR%
```

Notes
---
1. Compiler version.
During installation, Zuma configures the path
C:\c6xtools\bin...
for compiler vers. 2.0 and path
C:\ti\c6000\cgtools\bin
for complier vers. 3.01. In this installation, I have adjusted the path to use the vers. 3.01 compiler.

2. Autoexec.bat size limitation problem
After initial installation, the PC refused to boot. The problem was due to the changes made to autoexec.bat during the installation process. The solution to this problem was to limit the size of autoexec.bat by removing references to RTDX and to BIOS in the definition of C_dir.

3. Environment variable name changes.
After Zuma installation, environment variable C6x_c_dir was created. The developoment toolset (CodeWright, the TI compiler) failed to operate correctly until this was renamed c_dir.