Optical scanner galvanometer controller

1. Introduction

Laser beam scanning is almost ubiquitous in confocal and similar laser scanning microscopy instruments. It is most often achieved by using a pair of mirrors driven by galvanometer-type drivers. One of these galvanometers drives one of the mirrors back and forth in the horizontal direction (line scan), while the other drives the other mirror much more slowly in the vertical direction, thereby achieving a raster scan. The waveforms used to drive the mirrors have an essentially ramp-like shape, or similar, so as to ensure linear scans and a fast retrace time. We describe here a galvanometer controller unit, optimised to produce unidirectional scans according to principles described in companion documents “A method to overcome the effects of optical scanner hysteresis”, “A simple mount for scanner galvanometers” and optionally, “USB1 communications interface for controlling instruments”. We started development of this device during 2005 with a certain degree of concern, as it was then not obvious that our proposed method to eliminate the consequences of galvanometer hysteresis would actually prove effective in practice. In the event, it did and we continue to use this system to this day. Once the system was proven, we had intended to considerably simplify the construction of the system timing logic by using a Field Programmable Gate Array (FPGA), or indeed to investigate adaptation of commercial controllers to our scanning method. However, we never did! This is partially because the system described here, despite its complexity, is actually very simple and quick to construct, particularly as we had decided early on to design printed circuit boards for use in the instrument. The device proved to be very reliable and flexible and somehow or other we ended up by constructing six similar units, both for internal work and for that of our collaborators. We thus thought that others may benefit form this design and maybe, just maybe, we will updated it with more modern devices! We present reasonably detailed construction details and would be glad to assist should anyone wish to replicate the device; printed circuit board files can be supplied on request, as well as programming details. The one off total cost of the system is less than £3000, excluding the galvanometers and mirrors, so it is a reasonably cost-effective way of developing a very flexible laser scanning controller….should you require one of course!

We note that the analogue sections of the instrument are pretty general and could be applied to any other type of logic drive. Similarly, most of the effort goes towards constructing the high current power supply, an essential component of any fast scanning system which requires high peak currents to deal with the high scanner accelerations and rapid scanning speeds. The drivers for the galvanometers are standard commercial units and could be readily replaced with more modern devices, but these details, though important, do not detract from the basic approach. Nevertheless, if you have never constructed electronic equipment, this project is not for you. If you have, it will be pretty obvious how to modify the system to suit your needs

2. Ancillary equipment

Much of the microscopy work of our laboratory is associated with time-resolved fluorescence imaging. We routinely use signal (photon counting) acquisition cards made by Becker and Hickl (http://www.becker-hickl.de/) and our favourite is the SPC830 card. This scanner controller has been compatible with this card. Although complete scanning systems can in fact be purchased from B&H, the versatility of the device described here is somewhat greater and is desirable for development work.

The galvanometer scanners we use were readily available from General Scanning Inc, now GSI Lumonics. Although somewhat dated, they are still available in the UK from GSI Lumonics, Cosford Lane, Rugby,Warwickshire, CV21 1QN, Tel 01788 570321 and or from the parent company, see http://www.gsig.com/scanners/optical_spec.html. We use the MiniSax single axis driver in conjunction with VM-Series (VM1000) moving magnet galvanometers coupled to 10 mm scan mirrors. Similar, though somewhat updated devices can be obtained from Cambridge Technology (www.camtech.com), though we do not have experience with these systems. The GSI galvos have however proved to be extremely reliable (only two failures to date, from over 20 systems purchased over the years). We note however, that the drivers must be properly tuned according to the manufacturer's recommendations and that the correct tuning module for the scanner/mirror combination is fitted. Our design allows the various waveforms to be monitored in order to ensure correct operation.

The system we describe is fully programmable (i.e. there are no user controls) and adaptable for scan direction, scan channel reversal etc. and always maintains the 'correct' field of view, i.e. there is no image shift when changing speeds etc. Moreover, the scan can be restricted, thereby achieving image zooming in a logical way, as described in the note "A method to overcome the effects of optical scanner hysteresis". All the scanner driver functions are programmable through an I^2C bus, although the unit can be readily converted to allow control from the USB bus, as described in the accompanying note "USB1 communications interface for controlling instruments".

3. Scanner driver circuit description and printed circuit boards

The scanner driver is constructed in a ½ rack case, as shown in Figure 1. It is constructed in four compartments. The first houses two printed circuit boards, one dealing with the digital, or logic and timing sections, the second dealing with the scanner driver analogue systems, including digital-to analogue converters. The second section, behind the fan in Figure 1, houses the galvanometer drivers, hence the cooling. We find that a moderate degree of cooling helps with potential temperature rises during extended periods of operation and maintains component temperatures well below 50 deg.C. The third section, behind the on-off switch in the left panel in Figure 1, houses the DC power supply and finally, the rear panel modules take care of signals that may be required to interface monitoring or other instruments. Strictly speaking, the various connectors on these modules are not really required for normal operation, but we find them useful during setting up, eliminating the need for oscilloscope probes etc. and the consequent danger of 'expensive' shorts.

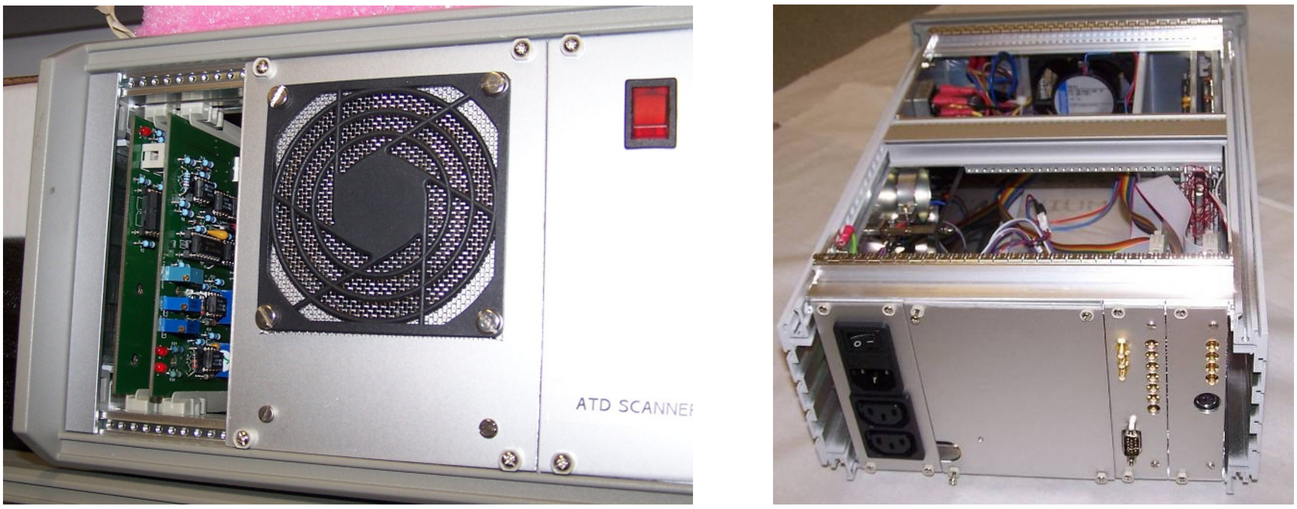


Figure 1. The scanner driver unit, from the front (left), with a panel removed to show the circuit boards and the rear. Interfcae signals are on the right of the right hand image and AC power connections are on the left. We provide two further IEC mains outputs for ancillary equipment as there are never enough mains sockets, even in the best laid out laboratory!

The electronics are constructed principally on two ‘Eurocard’ printed circuit boards, 160 x 100 mm. The first, logic board is shown in Figure 2. The details of the operation are presented elsewhere “A method to overcome the effects of optical scanner hysteresis”, but briefly, we use a 12 bit down-counter (3 x 74HCT191 counters) to define the line scan and a corresponding counter internal to a PIC microcontroller to define the vertical, or frame scan. The PIC microntroller (Microchip 16F877) controls other logic functions through an 8/12 bit bus, expanded using four latches (74HCT574s) and digital-to analogue converters, described later. The PIC communicated to the outside world using the I2C interface and can be on-board programmed using a 6 pin IDC socket, shown on the bottom left of Figure 2. ‘Glue’ logic and a series of D-type flip-flops ensure proper sequencing at the start and end of a scan and unbuffered signals defining the pixel, line and frame clocks are produced by this board. In our internal nomenclature, this board is called HTSCAN1 and it connects to the analogue board, designated HTSSCAN2 and HTSSCAN3 through wire-wrapped DIN 14612 connectors. Output signals are taken to the rear panel boards through a 14 way IDC ‘flat cable’ connector which plug directly into the rear of the DIN 14612 connectors.

There is of course no reason why this circuit could not be translated into FPGA code, but as indicated earlier, we just have not got round to doing this!

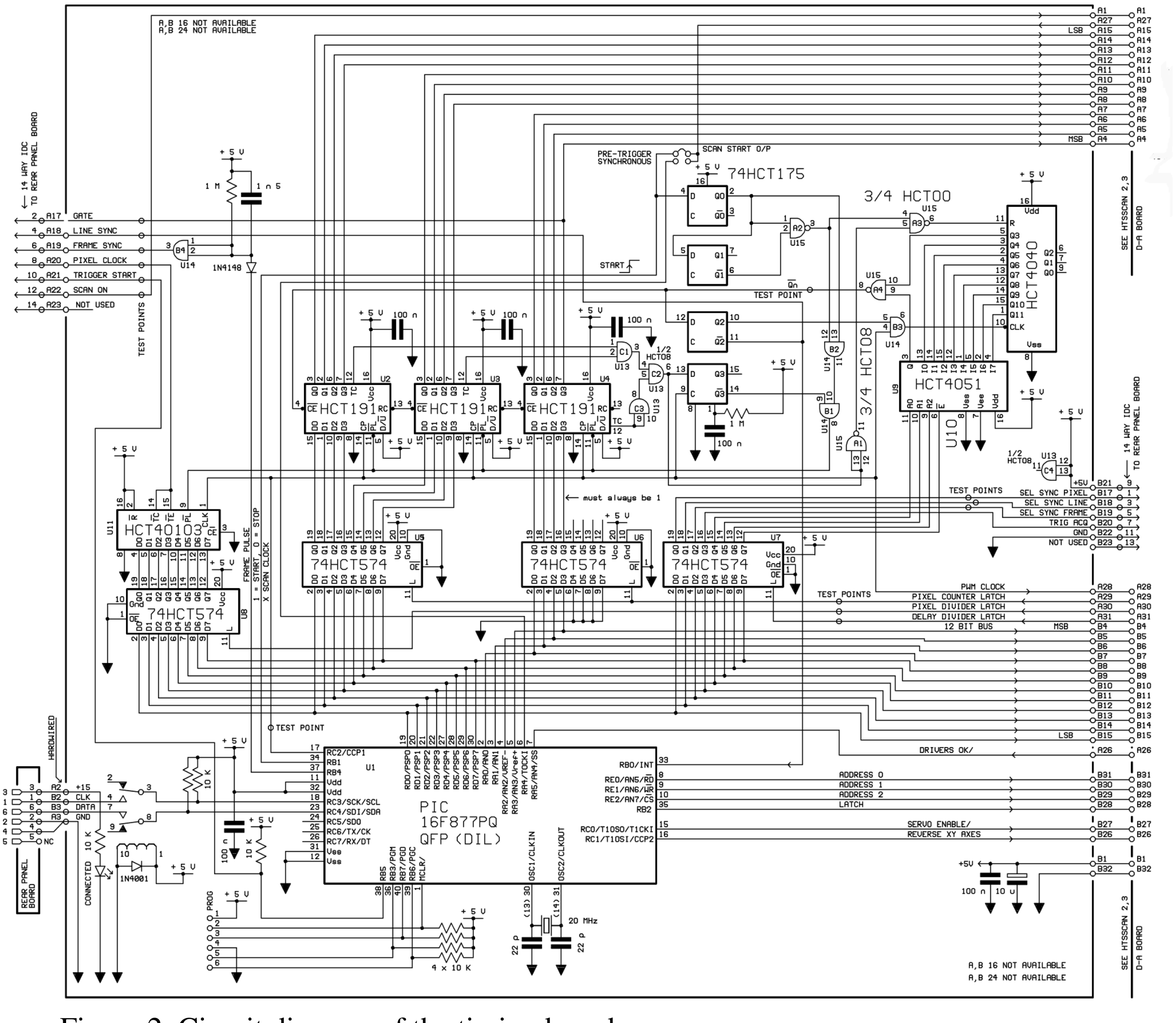


Figure 2. Circuit diagram of the timing board

The analogue board is shown in Figure 3. Here, we use two 12 bit digital-to-analogue converters (AD7845) to derive the basic line and frame scan waveforms, followed by two 8 bit multiplying digital to analogue converters (AD7524) to define the scan amplitude. Finally, two 8 bit converters define the scan offset, as required during zoomed image panning. In the case of the line scanning

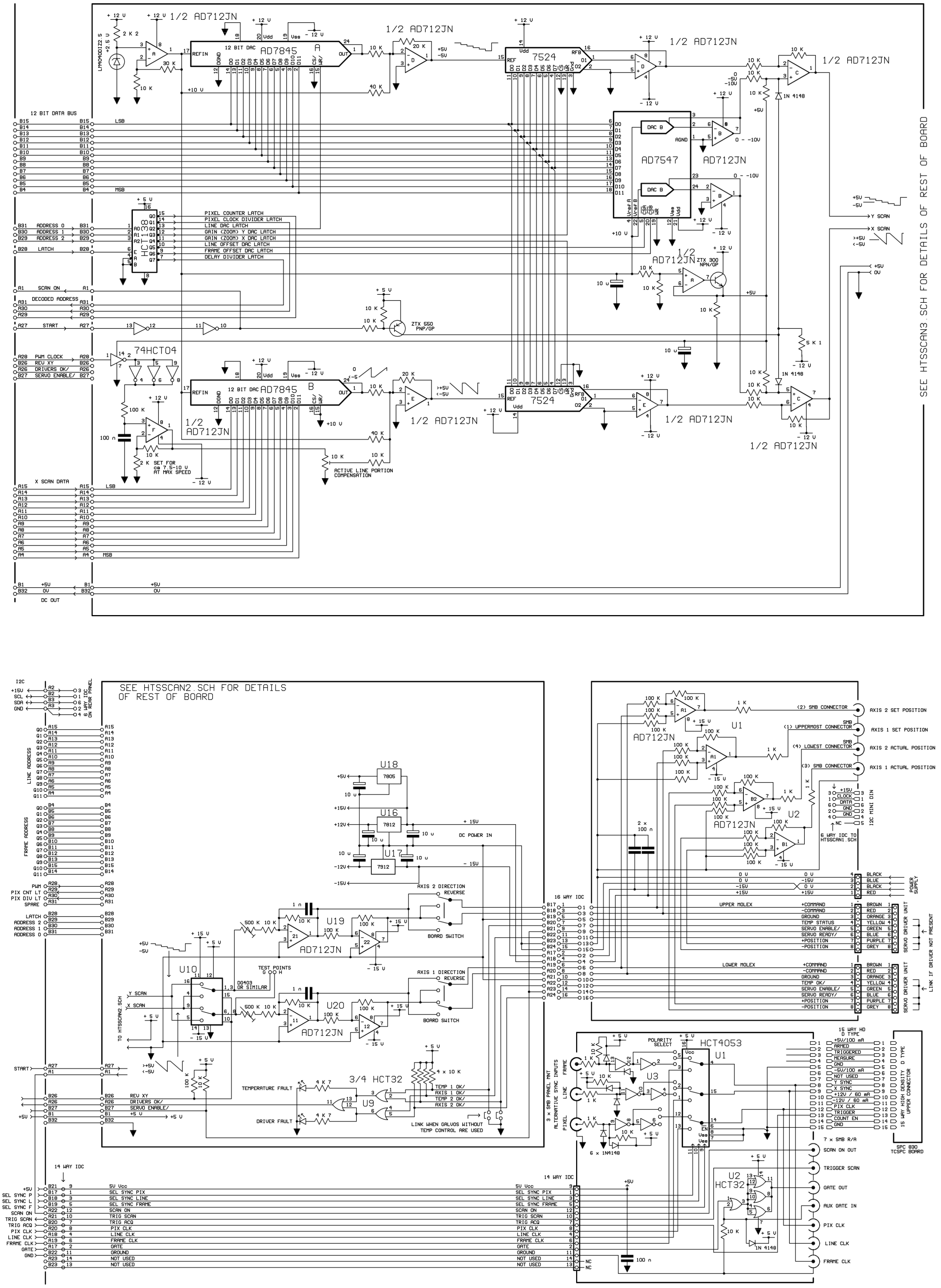


Figure 3. Circuit diagram of the analogue board and the rear panel boards.

signal, an additional, speed-dependant offset is injected, as described elsewhere, to compensate for the scanner hysteresis. The scan signals are fed to the lower part of Figure 3, where an analogue switch is used to reverse the vertical and horizontal scans if required, followed by a differential output circuit which allows the scan direction to be reversed, depending on the optical configuration, using a pair of DIP switches. This board also houses regulators to provide +5V, +12V and -12V as required by the rest of the circuits. The final differential scan signals are coupled to the scan drivers via one of the rear boards, shown at the upper right of Figure 3, though a 16 way IDC cable and a pair of 8 way SIL (Molex) connectors. This is done in order to make connection at the rear of the board relatively neat, since the MiniSax boards require a Molex-type input. This analogue rear board also contains differential input amplifiers which buffer the scan signals to and from the scanner drivers and make them available for monitoring purposes.

The bottom right part of Figure 3 shows the rear logic signal interface. Here, the pixel, line and frame clocks, as well as the triggering and gating signals are available on miniature coaxial connectors as well as on a B&H SPC830 board-specific high density D-type connector. We also provide a selector switch to allow other sources to trigger the data acquisition board.

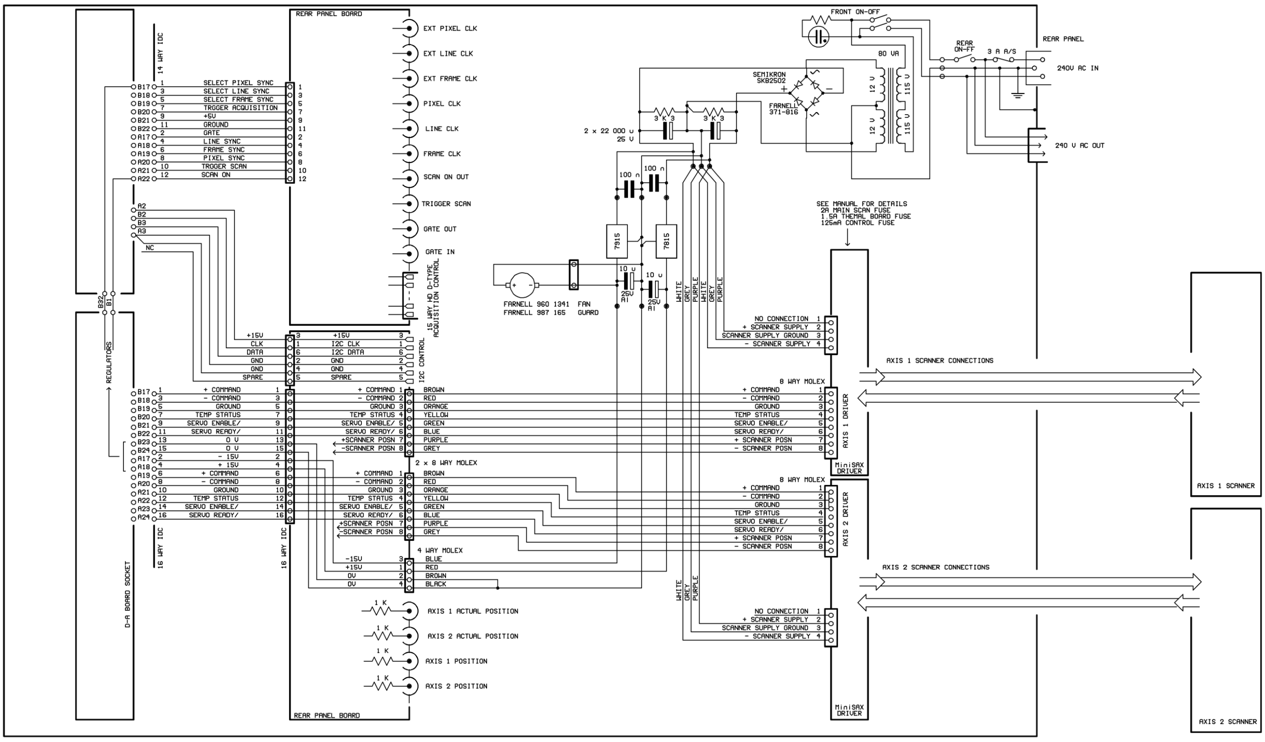
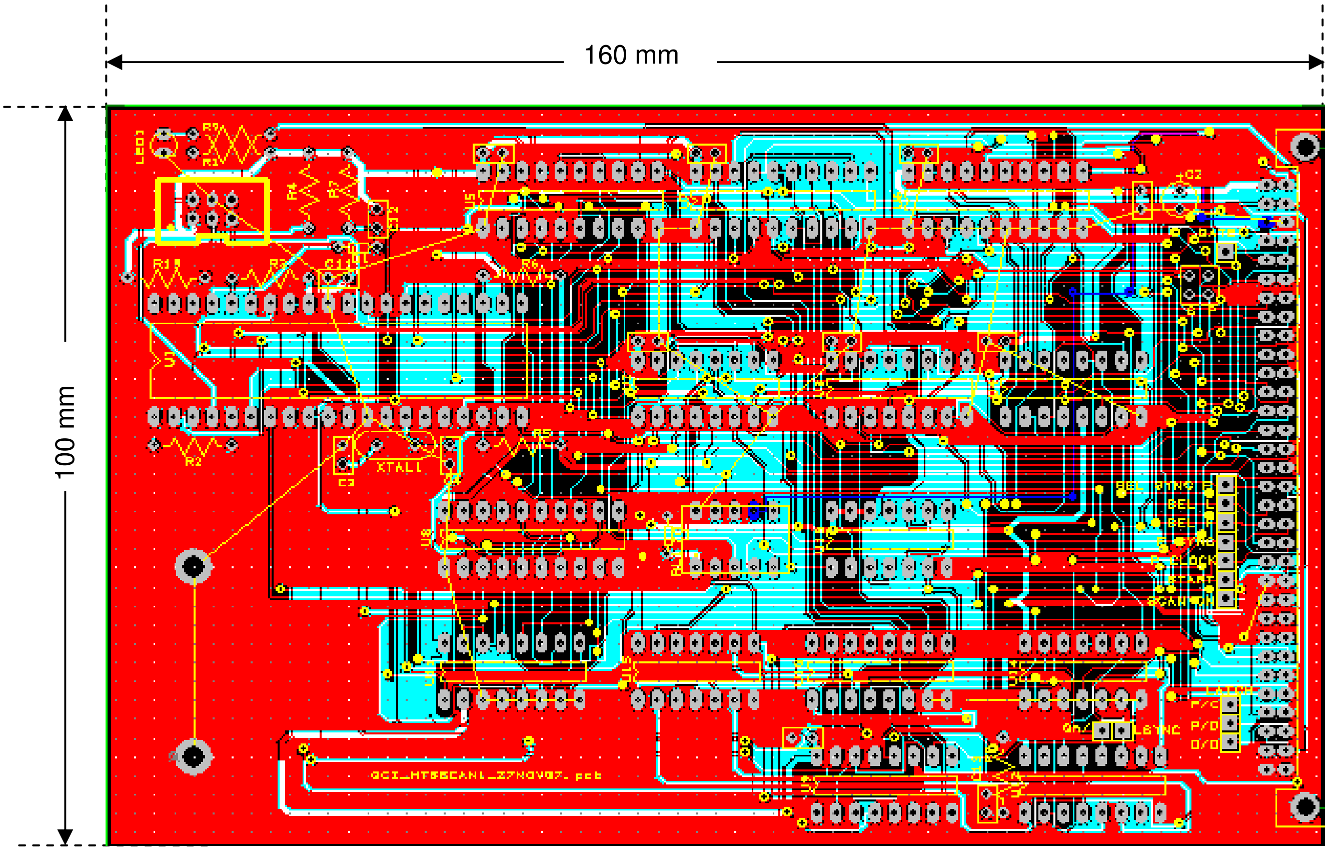
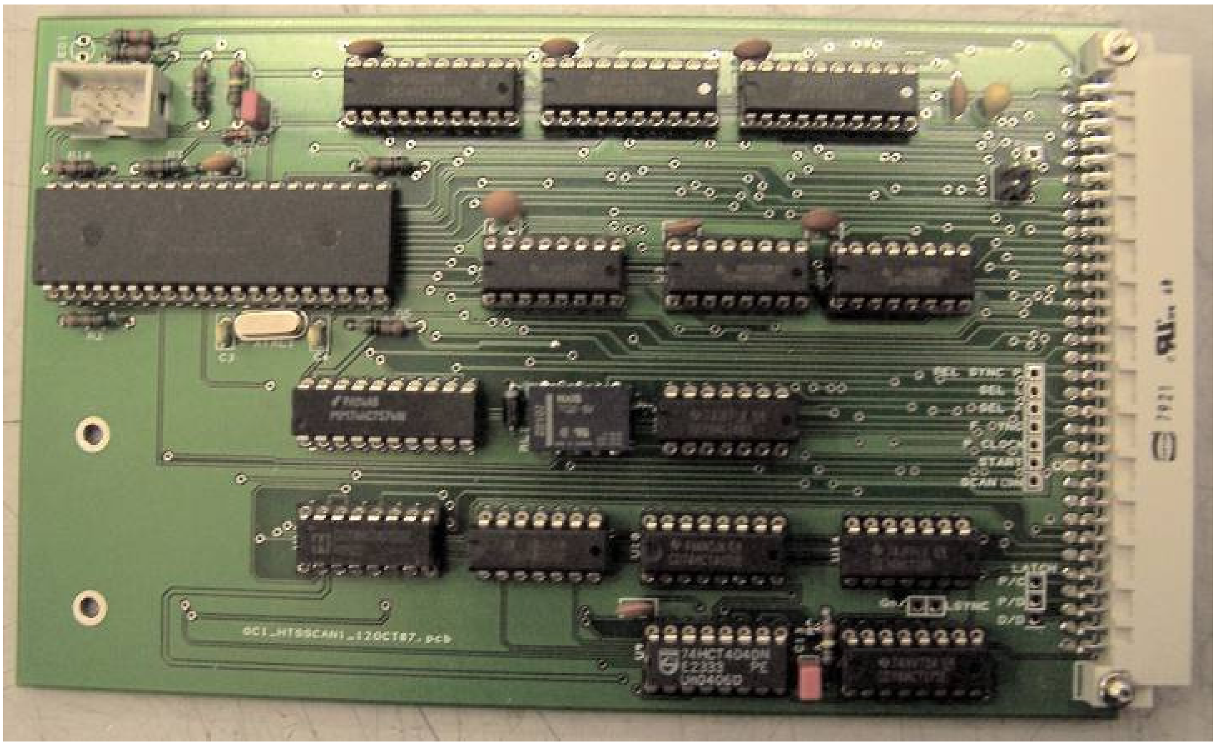


Figure 4. Circuit diagram of the power supply, board interconnections and scanner driver connections

The power supply and additional interconnections are shown in Figure 4. We use a simple bridge-rectified full-wave rectifier supply delivering ~±18V to the scanner drivers using a toroidal transformer and large reservoir capacitors to provide high peak currents. We also provide +/-15 V regulated supplies (further regulated down to ±12V locally on the boards in case these are required for additional applications. This power supply is constructed on an aluminium plate, as described later in the section dealing with mechanical drawings.

The circuit board layouts are shown on subsequent pages, Figures 5-8, and are self-expalnatory; we prpvide them here for completeness. We use PCB pool for board manufacture (http://www.pcb-pool.com/ppuk/info.html) and board assembly is straightforward.


160 mm
100 mm
XTAL1
SEL SYNC
SEL
START
SCAN ON
LATCH
P/C
P/D
D/D



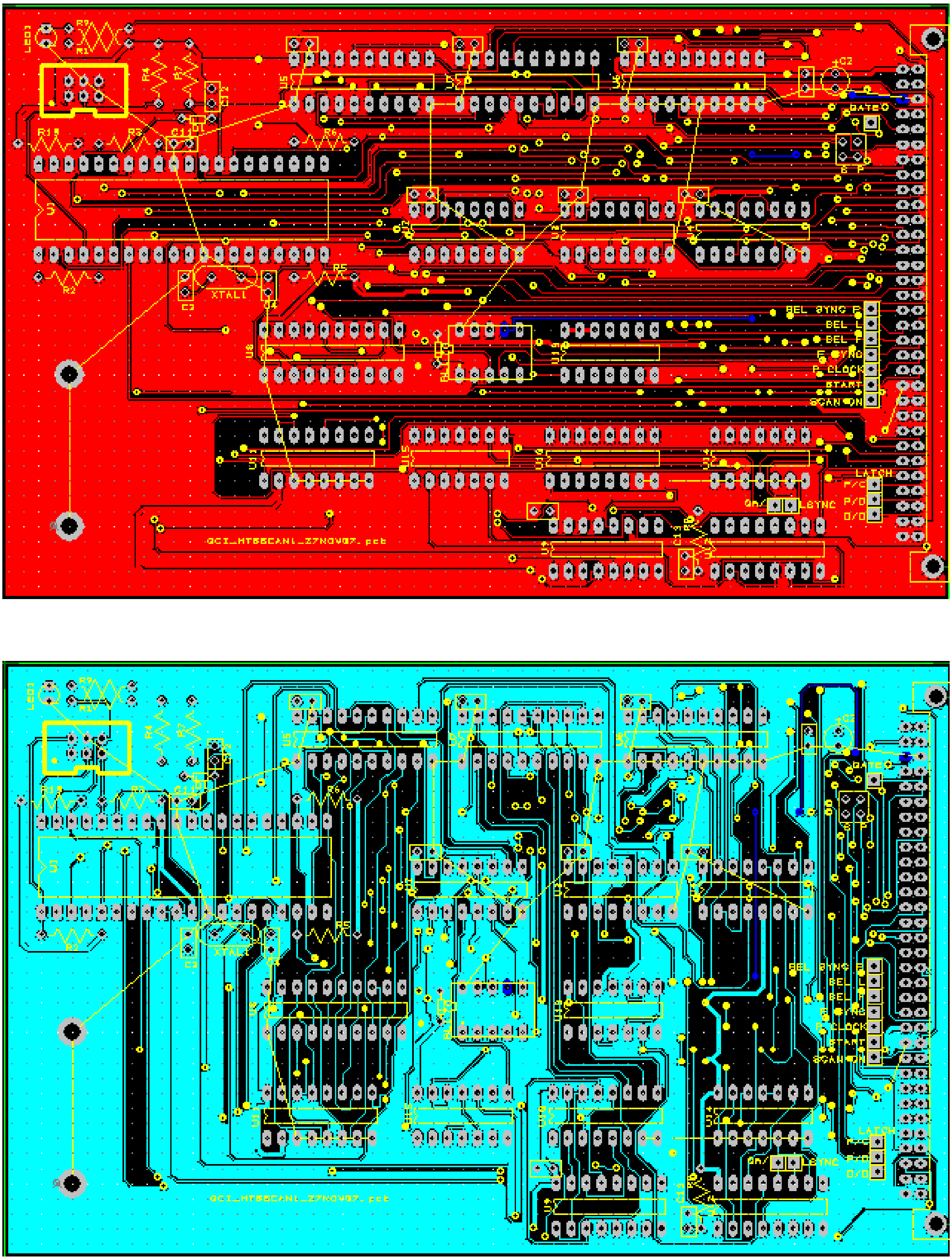
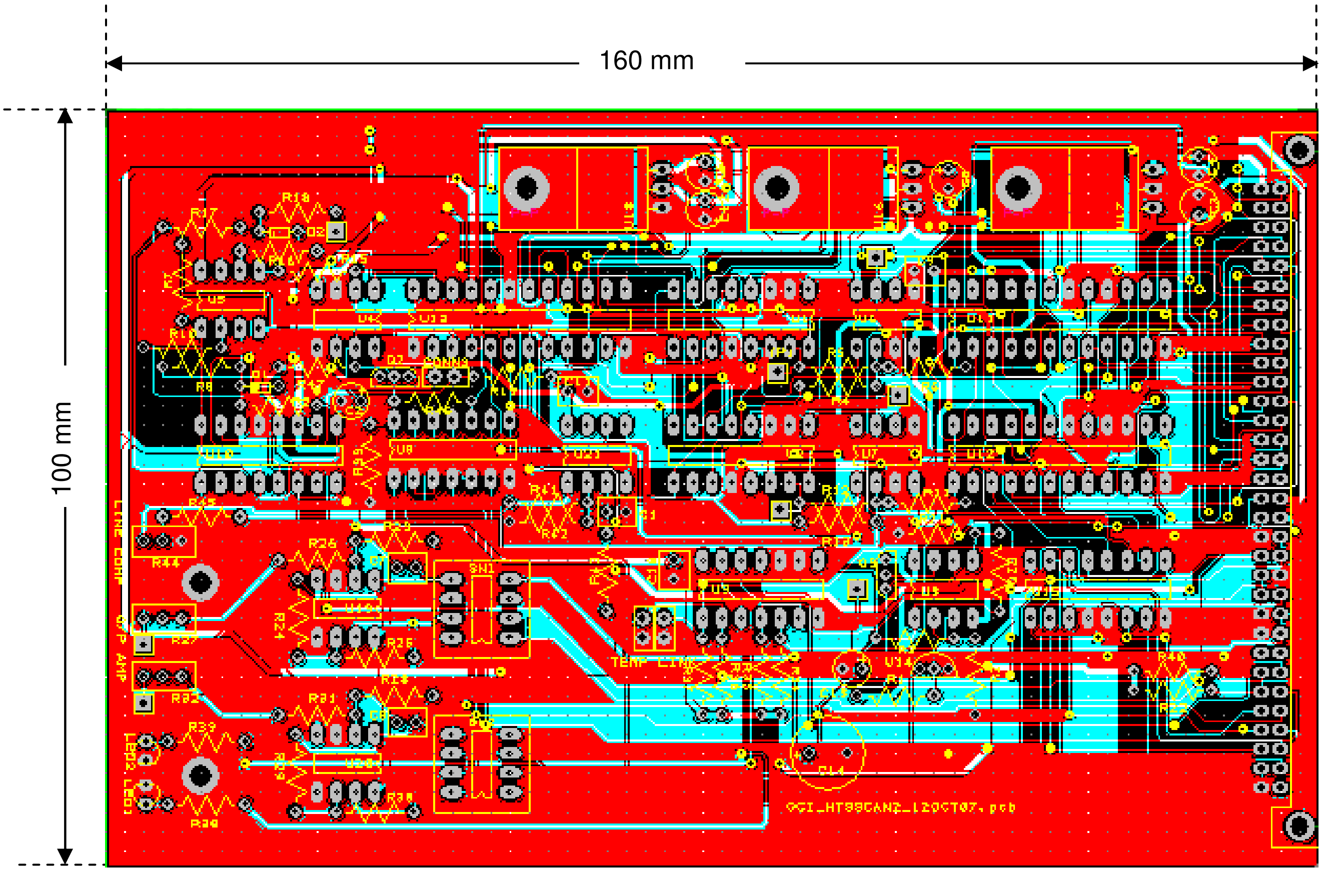
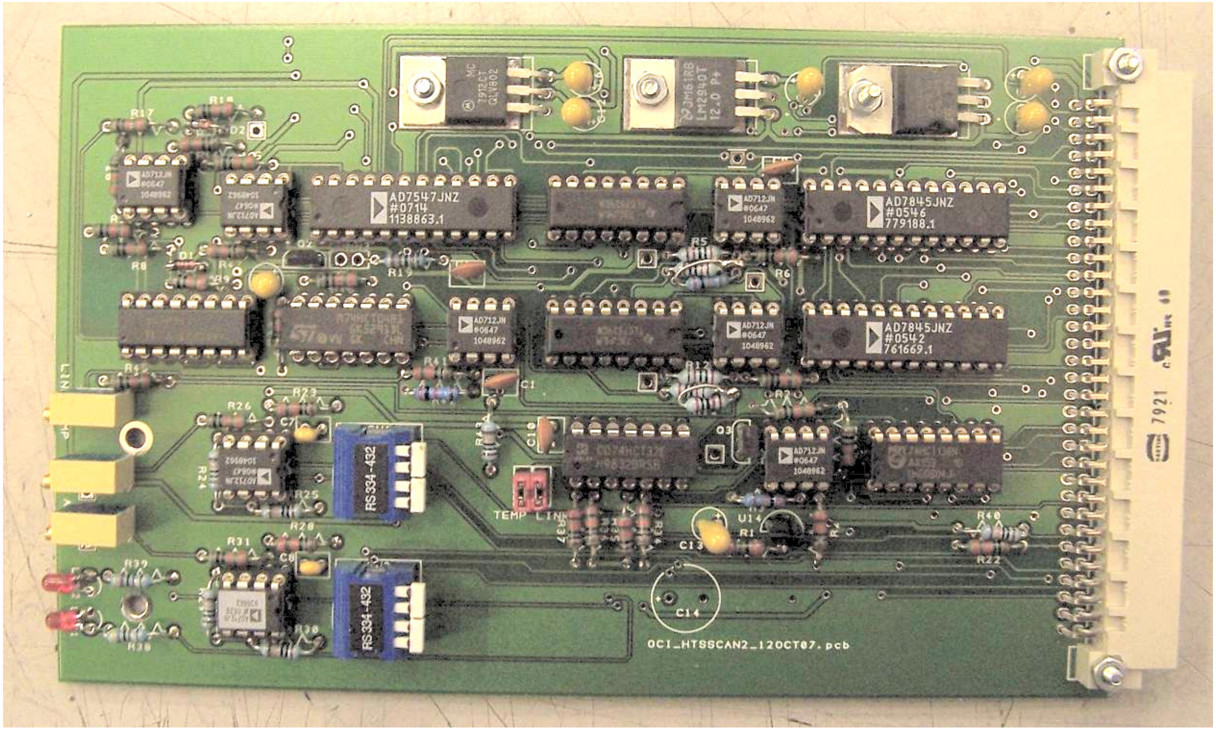


Figure 5. Double-sided printed circuit board layouts of the logic board.


160 mm
100 mm
OCI_HTSSCAN2_12OCT07.pcb


AD7547JNZ
AD7845JNZ
RS334-432
TEMP LIN
OCI_HTSSCAN2_12OCT07.pcb

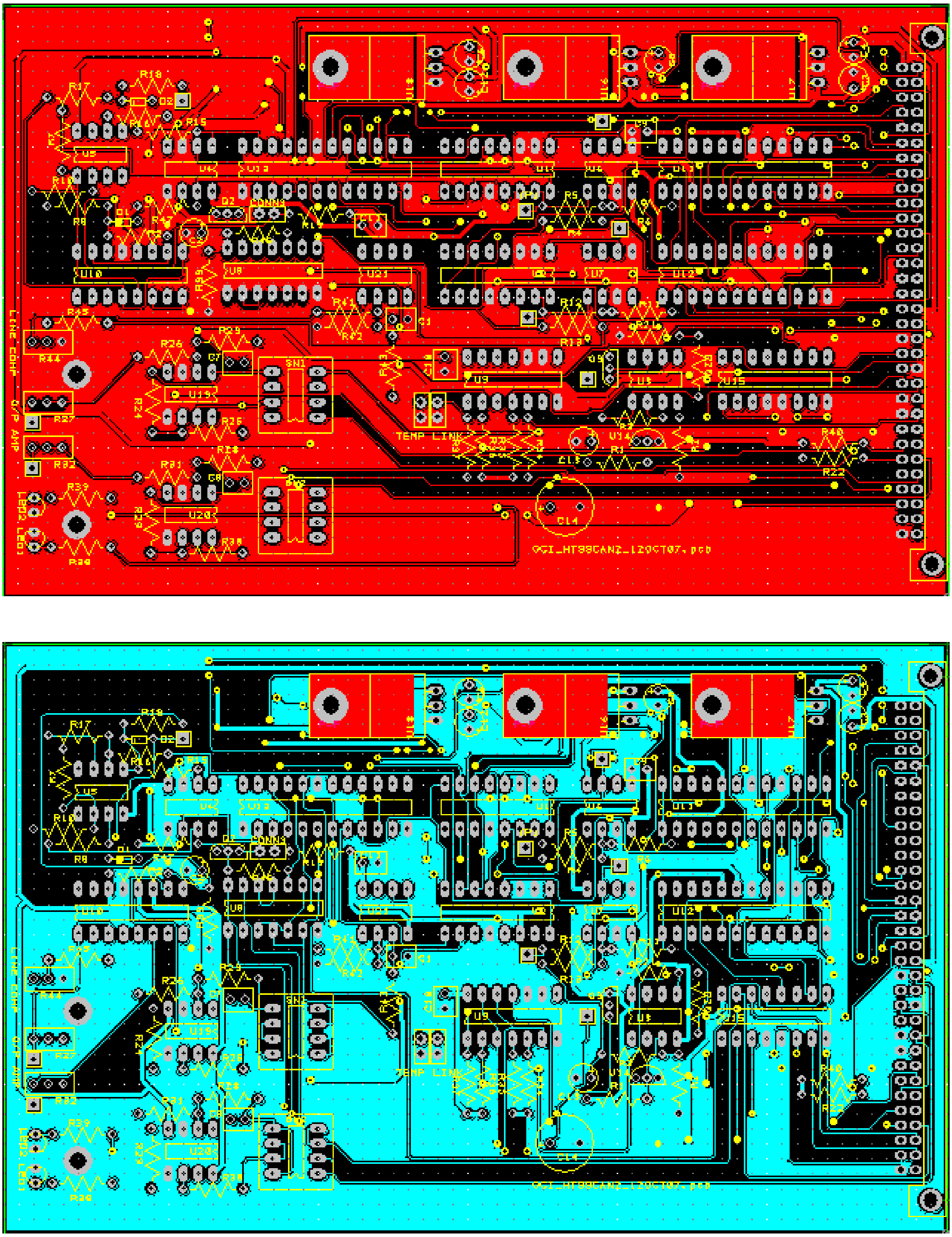


Figure 6. Double-sided printed circuit board layouts of the analogue board.

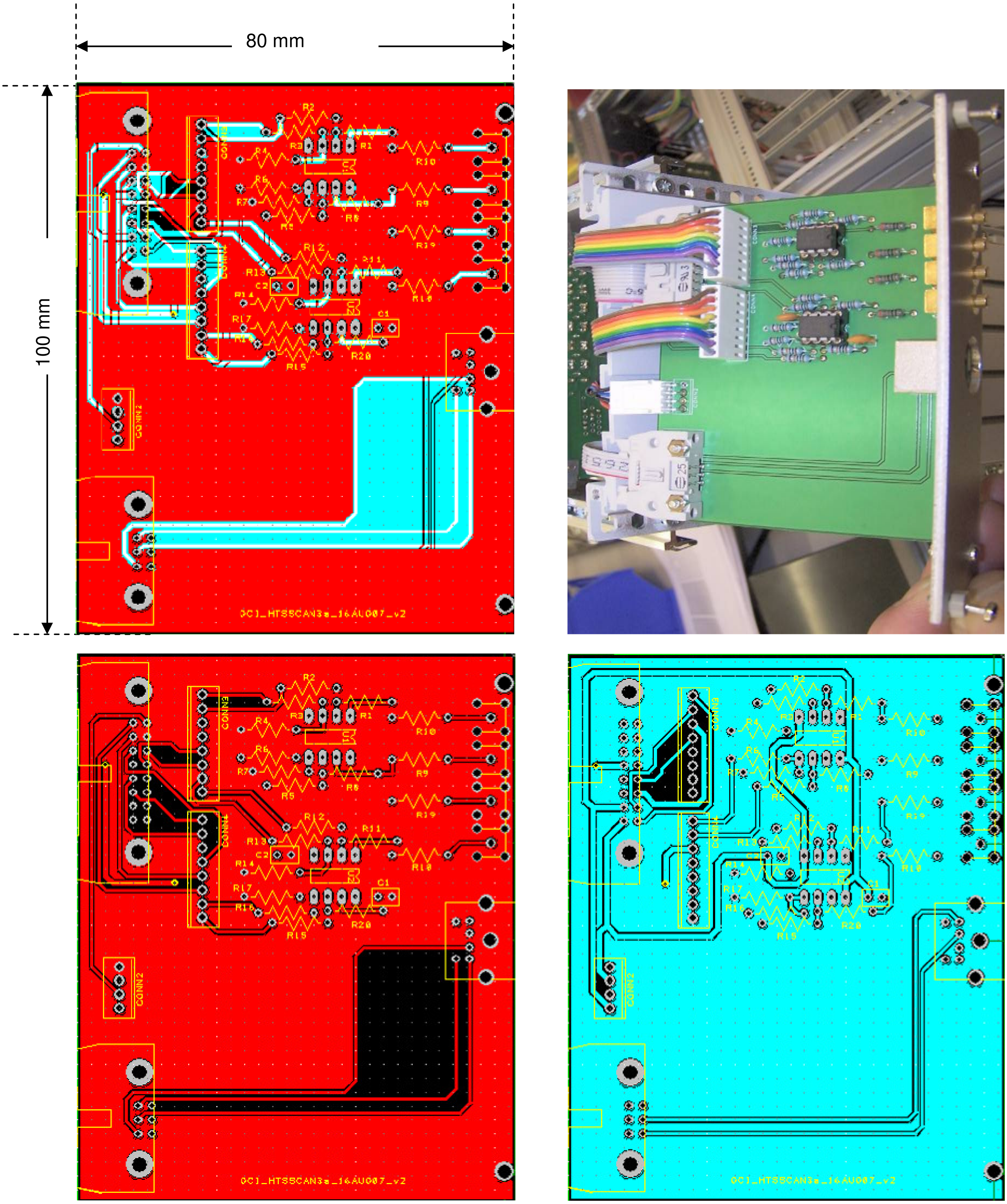


Figure 7. Double-sided printed circuit board layouts of the analogue output board.

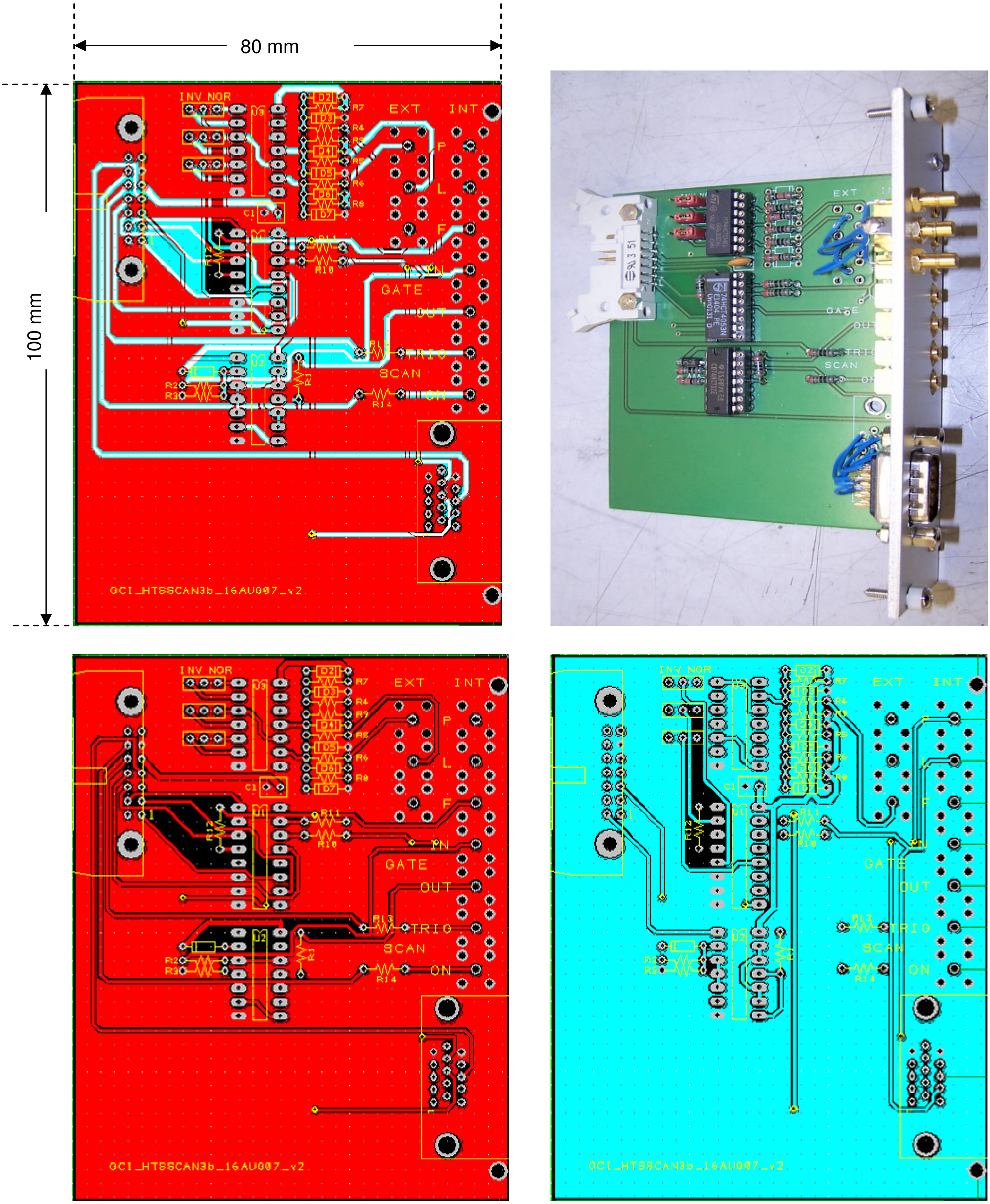


Figure 8. Double-sided printed circuit board layouts of the logic output board.

4. Construction details

We now present drawings of the mechanical components used in the assembly, starting with the power supply plate (Figure 9), a solid mounting plate for the scanner drivers (Figure 10), the rear panels (Figure 11) and the front panels (Figure 12).

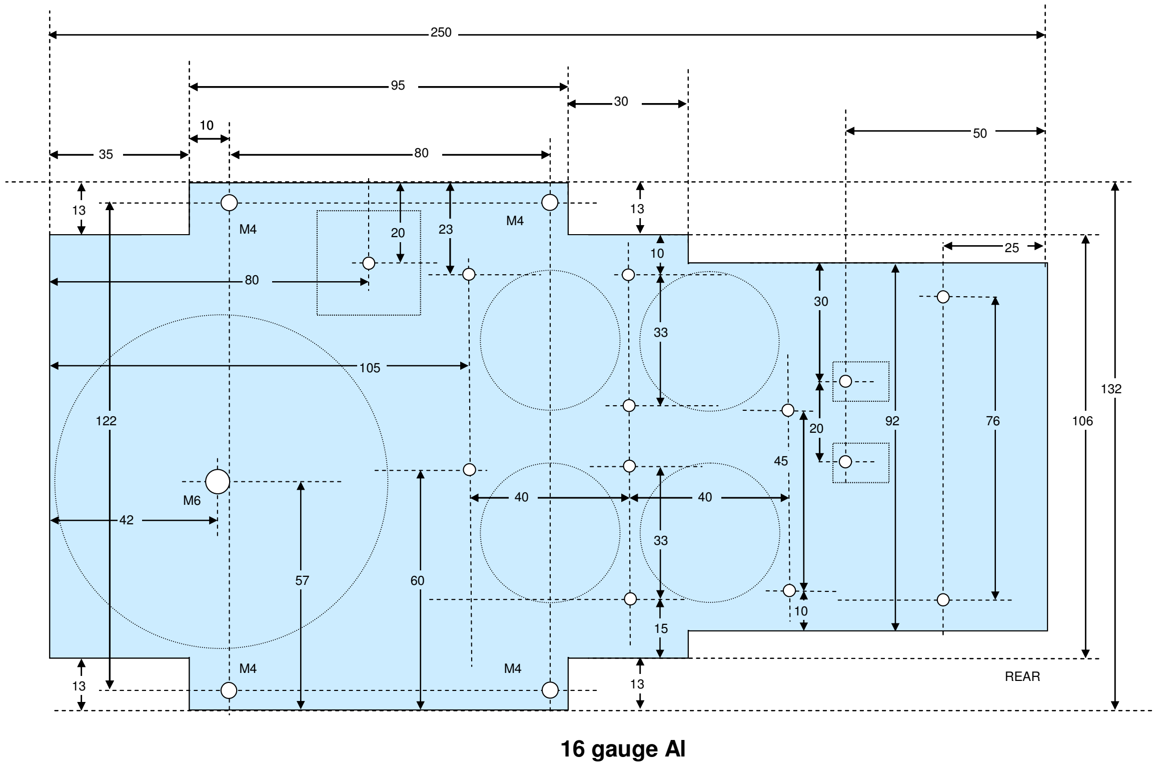


Figure 9. The power supply plate, capable of housing additional reservoir capacitors which may be required for ultra-fast scanning. A tagstrip is mounted on the right of the unit (76 mm hole separation) and connections soldered directly to this.

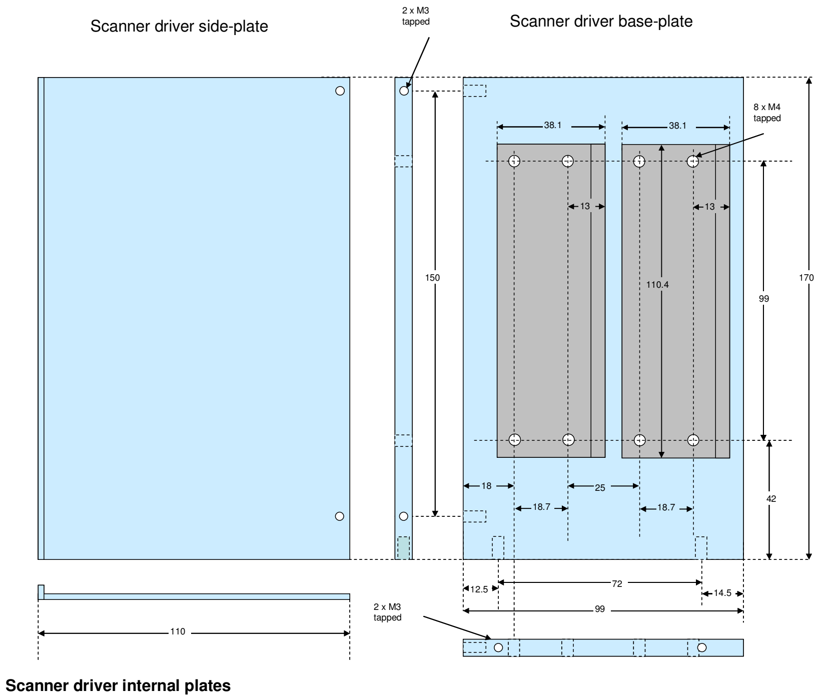


Figure 10. The scanner driver mounting plate (right), with a pair of drivers mounted vertically and a divider plate which screens the drivers form the boards.

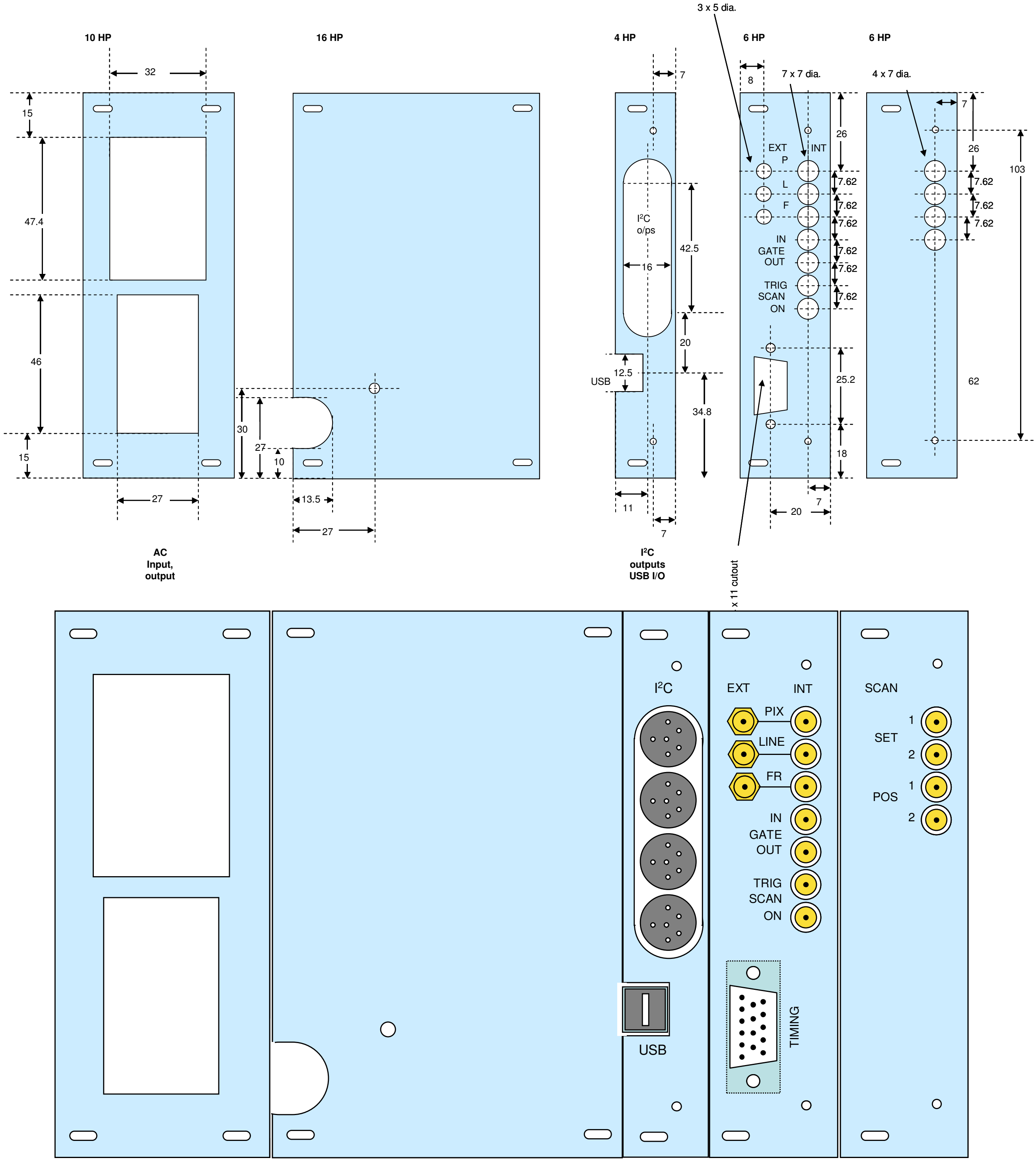


Figure 11. The rear panels, made from standard rack-mount components. The USB panel is only required if such an interface is needed. If I^{2}C-only control is required, the 16HP plate is replaced by a 20HP plate. Cables to the galvanometers are taken through the slot in this plate, and anchored with a ‘P’ clip.

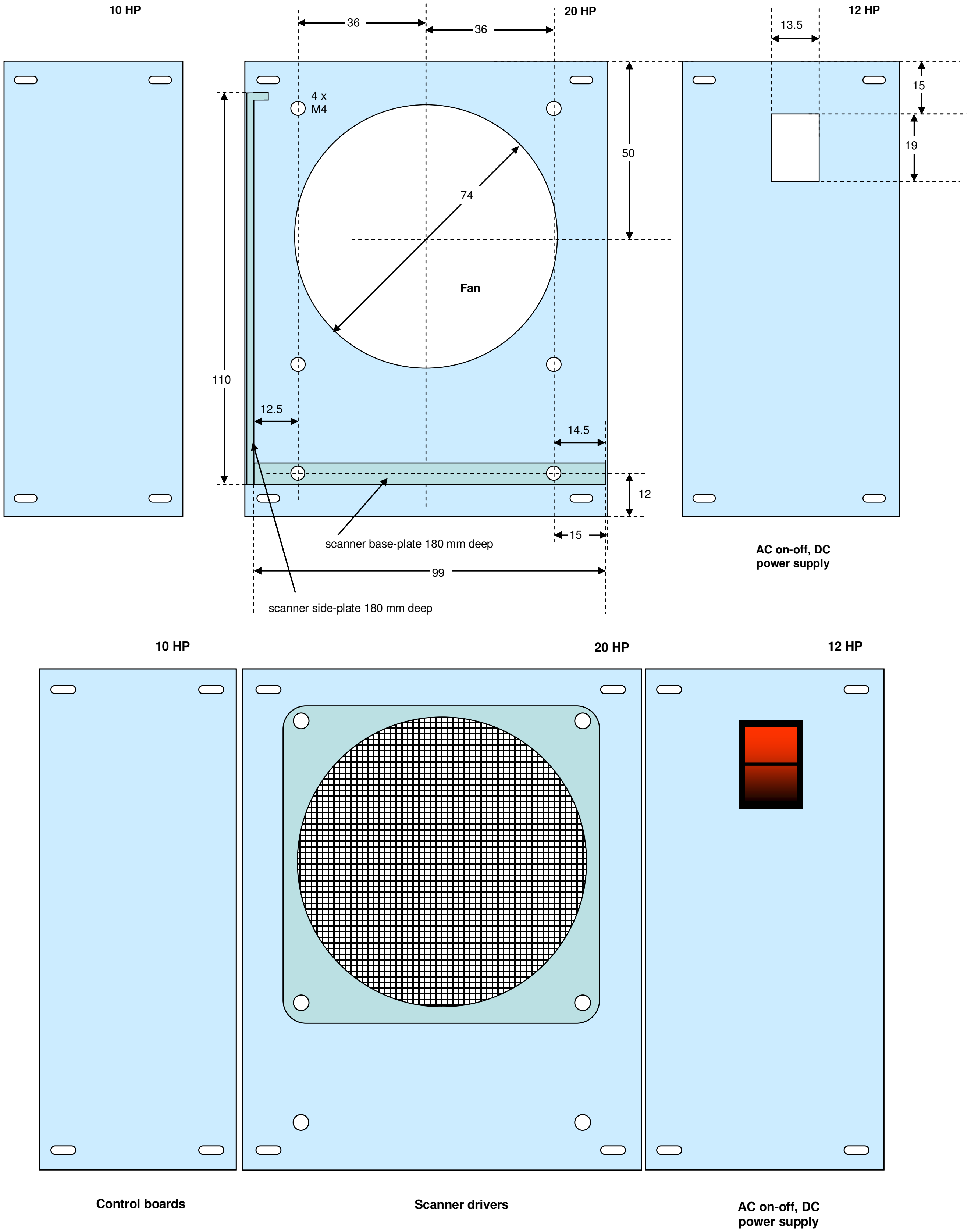


Figure 12. The front panels.

5. Component details

A comprehensive list of the components used in the construction of the scanner driver is provided here. We note that the component costs are unlikely to be correct, they represent 2008 prices and as we all know, the economy is not quite what it used to be! Nevertheless, they can be taken as a guide.

Key: **Blue = Electronic components**
Green = items made in GCI/ROB/Oxford electronics workshops, (printed circuit and electronic boards)
Purple = items made in GCI/ROB/Oxford mechanical workshops

Item	Description	Qty	Manufacturer part #	Supplier	Part number	£ each	£ total	
Scanner chassis								
Case								
Propac case	42HP half rack	1 off	RS / Schroff 10850017	RS	258-1264	£ 86.45	£ 86.45	
Front rails	To fit panels	4 off	RS / Schroff 20850265	RS	258-1882	£ 7.79	£ 31.16	
Rear rails	To fit board connectors	2 off	RS / Schroff 30819046	RS	258-2201	£ 6.69	£ 13.38	
Threaded insert	To fit case	4 off	RS / Schroff 30819636	RS	258-2138	£ 1.09	£ 4.36	
Trim	To fit case	1 kit	RS / Schroff 20850170	RS	258-1652	£ 7.59	£ 7.59	
Rail Screws	Bag of 10	1 off	RS / Schroff 21101416	RS	258-1911	£ 1.25	£ 1.25	
Plastic nipple	Bag of 100	1 off	RS / Schroff 21100-464	RS	542-4956	£ 5.25	£ 5.25	**SUB**
Panel screws	Bag of 100	1 off	RS / Schroff 21101-101	RS	484-8402	£ 9.45	£ 9.45	**TOTALS**
Board guides	Sold individually	6 off	Schroff 60817-103	Schroff	Not from RS	£ 0.36	£ 2.16	**£ 161.05**
Case and backplane assembly	--------------------------	1 off	---------------------	GCI/ROB	---------------------	£ 150.00	£ 150.00	**£ 150.00**

Item	Description	Qty	Manufacturer part #	Supplier	Part number	£ each	£ total	
Power supply and regulators								
HP12 panel	Front panel	1 off	RS / Schroff 20838116	RS	437-2012	£ 18.05	£ 3.61	
Front panel machining	--------------------------	1 off	--------------------------	GCI/ROB	------------------	£ 15	£ 15	
Side panel machining	--------------------------	1 off	--------------------------	GCI/ROB	------------------			
Rocker switch	DPST illuminated	1 off	19 x 13.5 cutout	Rapid	75-0300	£ 0.80	£ 0.80	
Smoothing capacitor	22000 µF 25V	2 off	Panasonic ECOS1EA223EA	Farnell	119-8544	£ 4.11	£ 8.22	
Bridge rectifier	25A 200V	1 off	Multicomp CM2502	Farnell	938-1198	£ 4.39	£ 4.39	
Transformer 80VA	2 x 12V	1 off	Multicomp MCTA080/12	Farnell	953-2706	£ 14.04	£ 14.04	
24V (12-28V) fan	80 mm diameter	1 off	Papst 8314L	Farnell	960-1341	£ 20.83	£ 20.83	
Fan guard	80 mm diameter	1 off	Multicomp MC0908G	Farnell	112-4771	£ 1.71	£ 1.71	**SUB**
Capacitor clips	35 mm dia	2/4 off	VISHAY BC Components	Farnell	118-7275	£ 0.96	£ 1.92	**TOTALS**
Regulator	+15V/3A	1 off	Fairchild MC78T15CT	RS	641-746	£ 1.43	£ 1.43	**£ 57.455**
Regulator	-15V/1.5A	1 off	ST LM7915ACV	RS	108-7145	£ 0.54	£ 0.54	**£ 85.00**
Power supply assembly construction	--------------------------	1 off	SSCAN4.SCH	GCI/ROB	---------------------	£ 85.00	£ 75.00	**£ 15.00**

Item	Description	Qty	Manufacturer / part #	Supplier	Part #	£ each	£ total	Note
Front board components								
4 bit programmable counter	74HCT191	3 off	Texas	RS	652-134	£ 1.16	£ 3.48	
8 bit latch	74HCT574	4 off	Texas	Farnell	110-5995	£ 0.61	£ 2.44	
Quad d-type flip-flop	74HCT175	1 off	Texas	Farnell	110-5984	£ 0.35	£ 0.35	
8 way selector	74HCT4051	1 off	Philips	Farnell	382-577	£ 0.64	£ 0.64	
12 bit counter	74HCT4040	1 off	Philips	Farnell	382-553	£ 0.43	£ 0.43	
Quad 2 input nand gate	74HCT00	1 off	Texas	Rapid	83-0010	£ 0.25	£ 0.25	
Quad 2 input and gate	74HCT08	2 off	Texas	Rapid	83-0014	£ 0.25	£ 0.50	
8-bit programmable down counter	74HC40103	1 off	Texas	Farnell	112-9243	£ 0.362	£ 0.362	
PIC processor	PIC16F877A-I/P	1 off	Microchip	Farnell	9761446	£ 4.07	£ 4.07	
20 MHz crystal	C-Mac LF A147K	1 off	C-Mac	Farnell	971-2879	£ 0.74	£ 0.74	
Small signal diode	1N4148	1 off	Multicomp	Farnell	956-5124	£ 0.011	£ 0.011	
"I2C connected" LED	Green	1 off	Vishay	Farnell	104-5460	£ 0.132	£ 0.132	
Decoupling capacitor	100 nF	4 off	Ceramic Y5V radial 2.5mm	Rapid	08-0275	£ 0.06	£ 0.24	
Decoupling capacitor	10 µF	1 off	Multicomp	Farnell	970-8448	£ 0.31	£ 0.31	
Oscillator capacitors	22 pF / 100 V	2 off	N150 2.5mm pitch, 4mm lead	RS	484-7724	£ 0.084	£ 0.168	
Resistors	10KΩ	8 off	Multicomp MF25	Farnell	934-1110	£ 0.021	£ 0.168	
Frame sync timing capacitor	1500 pF	1 off	LCR Components	Farnell	952-0104	£ 1.09	£ 1.09	
Start monostable capacitor	100 nF	1 off	Wima	Farnell	100-6004	£ 0.28	£ 0.28	
Timing resistors	1 MΩ	2 off	Multicomp MF25	Farnell	934-1137	£ 0.021	£ 0.042	
Pre-trigger link header	2+2 rows	1 off	No source	Rapid	22-0525	£ 0.04	£ 0.04	
Pre-trigger link jumper	Single link	1 off	FCI 68786-202LF	Farnell	109-7979	£ 0.08	£ 0.08	
Programming connector	6 way	1 off	Harting 0918 506 7324	Farnell	109-6984	£ 0.58	£ 0.58	
DIN 41612 socket b/plane	64 way A+B 2 row 13 mm	1 off	Harting 0902 264 6421	Rapid	19-2558	£ 3.00	£ 3.00	
DIN 41612 plug board	64 way A+B 2 row	1 off	Harting 0902 164 6921	Rapid	19-2554	£ 2.05	£ 2.05	
IC DIL socket 14 pin	0.3" turned pin	3 off	No source	Rapid	22-1721	£ 0.24	£ 0.72	
IC DIL socket 16 pin	0.3" turned pin	7 off	No source	Rapid	22-1722	£ 0.27	£ 2.89	
IC DIL socket 20 pin	0.3" turned pin	4 off	No source	Rapid	22-1724	£ 0.34	£ 1.36	**SUB**
IC DIL socket 40 pin	0.3" turned pin	1 off	No source	Rapid	22-1730	£ 0.68	£ 0.68	**TOTALS**
Printed circuit board		1 off	GCI_HTSSCAN1_16AUG07_v3.PCB	PCB pool		£ 36.00	£ 36.00	**£ 63.10**
Electronics board		1 off	SSCAN1.SCH	GCI/ROB		£ 100.00	£ 100.00	**£ 100.00**

Item	Description	Qty	Manufacturer / part #	Supplier	Part #	£ each	£ total	Note
Regulator +12V/1A	MC78T12CV	1 off	ST Microelectronics	Rapid	47-3292	£ 0.32	£ 0.32	
Regulator -12V/1A	LM7912CV	1 off	ST Microelectronics	Rapid	47-3299	£ 0.35	£ 0.35	
Regulator +5V/1A	MC78T05CV	1 off	ST Microelectronics	Rapid	47-3290	£ 0.25	£ 0.25	
12 bit DAC	AD7845 JNZ	2 off	Analog Devices	Farnell	960-4588	£ 12.35	£ 24.70	
8 bit DAC	AD7524 JNZ	2 off	Analog Devices	Farnell	960-5401	£ 7.55	£ 15.10	
Dual 12 bit DAC	AD7547JNZ	1 off	Analog Devices	Farnell	960-4570	£ 27.86	£ 27.86	
Hex inverter / PWM driver	74HCT04	1 off	Texas	Rapid	83-0012	£ 0.25	£ 0.25	
1-of-8 decoder	74HCT138	1 off	Texas	Rapid	83-0022	£ 0.28	£ 0.28	
2.5 V reference	LM4040DIZ-2.5/NOPB	1 off	National Semiconductor	Farnell	115-0259	£ 0.393	£ 0.393	
NPN reference buffer	ZTX450	1 off	Zetex	Farnell	952-5513	£ 0.27	£ 0.27	
PNP gate buffer	ZTX550	1 off	Zetex	Farnell	952-5548	£ 0.33	£ 0.33	
Dual 2p2w analogue switch	DG403DJ-E3	1 off	Vishay-Siliconix	Farnell	1077116	£ 2.33	£ 2.33	
Quad 2 input OR gate	74HCT32	1 off	Philips	Farnell	381-871	£ 0.37	£ 0.37	
Small signal diode	1N4148	2 off	Multicomp	Farnell	956-5124	£ 0.011	£ 0.022	
Dual opamp	AD712JNZ	8 off	Analog Devices	Rapid	82-0455	£ 2.68	£ 21.44	
"Temperature fault" LED	Red	1 off	Vishay	Farnell	104-5504	£ 0.165	£ 0.165	
"Driver fault" LED	Yellow	1 off	Vishay	Farnell	104-5467	£ 0.250	£ 0.250	
Reference decoupling capacitor	10 µF	1 off	Multicomp CB1E106M2ICB	Farnell	970-8448	£ 0.31	£ 0.31	
PWM compensation capacitor	100 nF	1 off	Ceramic Y5V radial 2.5mm	Rapid	08-0275	£ 0.06	£ 0.06	
Decoupling capacitors	10 µF	5 off	Multicomp	Farnell	970-8448	£ 0.31	£ 1.55	
Output settling capacitors	1 nF	2 off	Wima	Farnell	100-6008	£ 0.179	£ 0.358	
Reference resistor	2.2 KΩ	1 off	Multicomp MF25	Farnell	934-1536	£ 0.021	£ 0.021	
Gain resistors	10 KΩ	15 off	Multicomp MF25	Farnell	934-1110	£ 0.021	£ 0.315	
Gain resistors	100 KΩ	4 off	Multicomp MF25	Farnell	934-1129	£ 0.021	£ 0.084	
Gain resistors	20 KΩ	6 off	Multicomp MF25	Farnell	934-1498	£ 0.021	£ 0.126	
Pull up resistors	10 KΩ	6 off	Multicomp MF25	Farnell	934-1110	£ 0.021	£ 0.126	
Pull-up resistor	100 KΩ	1 off	Multicomp MF25	Farnell	934-1129	£ 0.021	£ 0.084	
LED resistors	4.7 KΩ	2 off	Multicomp MF25	Farnell	934-1951	£ 0.021	£ 0.042	
Compensation gain resistor	100 KΩ	1 off	Multicomp MF25	Farnell	934-1129	£ 0.021	£ 0.084	
Compensation gain resistor	10 KΩ	1 off	Multicomp MF25	Farnell	934-1110	£ 0.021	£ 0.021	
Compensation gain resistor	2 KΩ	1 off	Multicomp MF25	Farnell	934-1480	£ 0.021	£ 0.021	

Gating resistor	5.1 KΩ	1 off	Multicomp MF25	Farnell	934-2010	£ 0.021	£ 0.021	
Line amplitude span preset	500 KΩ	1 off	Vishay 64X-504	Farnell	960-8540	£ 1.21	£ 1.21	
Frame amplitude span preset	500 KΩ	1 off	Vishay 64X-504	Farnell	960-8540	£ 1.21	£ 1.21	
Line compensation preset	10 KΩ	1 off	Vishay 64X-103	Farnell	960-8494	£ 1.21	£ 1.21	
DIL switch	2x SPDT, linked	1 off	(Tyco?) NP2	RS	334-432	£ 4.70	£ 4.70	
DIN 41612 socket b/plane	64 way A+B 2 row 13 mm	1 off	Harting 0902 264 6421	Rapid	19-2558	£ 3.00	£ 3.00	
DIN 41612 plug board	64 way A+B 2 row	1 off	Harting 0902 164 6921	Rapid	19-2554	£ 2.05	£ 2.05	
IC DIL socket 8 pin	0.3” turned pin	8 off	No source	Rapid	22-1720	£ 0.14	£ 1.12	
IC DIL socket 14 pin	0.3” turned pin	2 off	No source	Rapid	22-1721	£ 0.24	£ 0.48	
IC DIL socket 16 pin	0.3” turned pin	4 off	No source	Rapid	22-1722	£ 0.27	£ 1.08	
IC DIL socket 24 pin	0.3” turned pin	3 off	No source	Rapid	22-1725	£ 0.40	£ 1.20	
Temperature detect link header	2+2 rows	1 off	No source	Rapid	22-0525	£ 0.04	£ 0.04	**SUB**
Temperature detect link jumper	Single link	2 off	FCI 68786-202LF	Farnell;	109-7979	£ 0.08	£ 0.16	**TOTALS**
Printed circuit board		1 off	GCI_HTSSCAN2_12OCT07.PCB	PCB pool		£ 36.00	£ 36.00	**£ 150.95**
Electronics board		1 off	SSCAN2.SCH, SSCAN3.SCH	GCI/ROB		£ 100.00	£ 100.00	**£ 100.00**

Item	**Description**	**Qty**	**Manufacturer / part #**	**Supplier**	**Part #**	**£ each**	**£ total**	**Note**
Rear board component								
Scanner drive amplifiers								
HP10 panel (1/2 42HP panel)	Rear panel	1 off	RS / Schroff 20838114	RS	437-1996	1/5 £ 16.35	£ 3.27	
HP6 panel	Rear panel	1 off	RS / Schroff 20838110	RS	437-1968	1/5 £ 12.25	£ 2.45	
Panel machining		1 off		GCI/ROB			£ 20	
Power supply								
HP10 panel	Rear panel	1 off	RS / Schroff 20838114	RS	437-1996	£ 16.35	£ 3.27	
Panel machining		1 off		GCI/ROB	---------------------	£ 40	£ 40	**SUB**
AC output connector	2 way IEC socket	1 off	RC 32 x 47 cutout	Rapid	23-3107	£ 0.95	£ 0.95	**TOTALS**
AC input connector	Fused/switched IEC	1 off	RC 46 x 27 cutout	Rapid	23-3209	£ 1.45	£ 1.45	**£ 8.32**
Insulating boots	Cover for AC inputs	2 off	No source	Rapid	23-0357	£ 0.55	£ 0.55	**£ 50.00**
Side panel assembly / wiring	--------------------------	1 off	SSCAN4.SCH	GCI/ROB	---------------------	£ 50.00	£ 50.00	**£ 60**

Item	**Description**	**Qty**	**Manufacturer / part #**	**Supplier**	**Part #**	**£ each**	**£ total**	**Note**
USB interface rear panel								
HP4 panel	Rear panel	1 off	RS/ Schroff 20838108	RS	437-1946	1/5 £ 10.85	£ 2.17	
Panel machining	-----------------------------------	1 off	Surrey end-station system modules .p	GCI/ROB	-----------------	£ 15.00	£ 15.00	
USB interface	DLP245PB	1 off	Future Technology Devices Intl. Ltd.	FTD	XXXXXXXX	£ 54.00	£ 54.00	
Mini DIN socket	6 way board mount	4 off	Protech LNB series	Rapid	20-0690	£ 0.49	£ 1.96	
IDC cable socket	6 way	2 off	Harting 0918 506 7813	Farnell	109-7021	£ 0.60	£ 1.20	
IDC header / pcb plug	6 way R/A	2 off	Harting 0918 506 7913	Farnell	110-6744	£ 0.67	£ 1.34	
DC power board plug	6 way R/A	1 off	Molex 22-05-7068	Farnell	973-1644	£ 0.71	£ 0.71	**SUB**
DC power header	6 way shell	1 off	Molex 22-01-2065	Farnell	143-129	£ 0.22	£ 0.22	**TOTALS**
I^2C pull-up resistors	10 kΩ	2 off	Multicomp MF25	Farnell	934-1110	£ 0.021	£ 0.042	**£ 79.70**
Printed circuit board		1 off	GCI_HTSCONT1_04JUN08.PCB	PCB pool		£ 18.00	£ 18.00	**£ 25.00**
Electronics board construction	PCB USB interface board	1 off	SCONT1.SCH	GCI/ROB	-----------------	£ 25.00	£ 25.00	**£ 15.00**

Item	**Description**	**Qty**	**Manufacturer / part #**	**Supplier**	**Part #**	**£ each**	**£ total**	**Note**
HP6 panel	Rear panel	1 off	RS / Schroff 20838110	RS	437-1968	1/5 £ 12.25	£ 2.45	
Panel machining		1 off	--------------------------------	GCI/ROB	---------------------	£ 20.00	£ 20.00	
Position board decoupling caps	100 nF capacitor	2 off	Ceramic Y5V radial 2.5mm	Rapid	08-0275	£ 0.06	£ 0.12	
Position board output resistors	1 KΩ	4 off	Multicomp MF25	Farnell	934-1102	£ 0.021	£ 0.084	
Position board gain resistors	100 KΩ	16 off	Multicomp MF25	Farnell	934-1129	£ 0.021	£ 0.336	
Dual opamp	AD712JNZ	2 off	Analog Devices	Rapid	82-0455	£ 2.68	£ 5.36	
Pix/frame/line inputs	SMB chassis mount	3 off	Tyco/Greenpar 1-1337479-0	Farnell	105-6343	£ 2.03	£ 6.09	
Position output connectors	R/A board mount SMB	3 off	Multicomp 24-12-2-TGG	Rapid	16-1508	£ 2.05	£ 6.15	
Timing i/o connectors	R/A board mount SMB	3 off	Multicomp 24-12-2-TGG	Rapid	16-1508	£ 2.05	£ 6.15	
Timing output D-type plug	15 way High Density	1 off	McMurdo HDE15PTD	Farnell	107-1808	£ 2.02	£ 2.02	
Screw-lock assembly	1 pair per connector 8 mm	3 off	Chin Nan Prec’n Electr’ics 4-40 UNC	Rapid	15-0365	£ 0.34	£ 0.34	
Scanner control connector	8-way shell	4 off	Molex 22-01-2085	Farnell	143-130	£ 0.25	£ 1.00	
Scanner control connector	8-way header	4 off	Molex 22-27-2081	Farnell	973-1180	£ 0.80	£ 3.20	
IC DIL socket 8 pin	0.3” turned pin	2 off	No source	Rapid	22-1720	£ 0.14	£ 0.28	**SUB**
IDC cable socket	16 way	2 off	Harting 0918 516 7813	Farnell	109-7025	£ 0.87	£ 1.74	**TOTALS**
IDC header / pcb plug	16 way R/A	1 off	Harting 0918 516 7913	Farnell	110-6747	£ 0.94	£ 0.94	**£ 36.26**
Printed circuit board		1 off	GCI_HTSSCAN3a_16AUG07.PCB	PCB pool		£ 18.00	£ 18.00	**£ 25.00**
Electronics board	--------------------------	1 off	SSCAN3.SCH	GCI/ROB	---------------------	£25.00	£25.00	**£ 20.00**

Item	**Description**	**Qty**	**Manufacturer / part #**	**Supplier**	**Part #**	**£ each**	**£ total**	**Note**
HP6 panel	Rear panel	1 off	RS / Schroff 20838110	RS	437-1968	1/5 £ 12.25	£ 2.45	
Panel machining	----------------------	1 off		GCI/ROB		£ 20.00	£ 20.00	
Quad 2 input OR gate	74HCT32	1 off	Texas	Farnell	110-5968	£ 0.28	£ 0.28	
Triple selector switch	74HCT4053	1 off	Philips	Farnell	382-590	£0.48	£ 0.48	
Decoupling capacitor	100 nF capacitor	1 off	Ceramic Y5V radial 2.5mm	Rapid	08-0275	£ 0.06	£ 0.06	
Resistor	10 KΩ	1 off	Multicomp MF25	Farnell	934-1110	£ 0.021	£ 0.021	
Timing i/o connectors	R/A board mount SMB	4 off	Multicomp 24-12-2-TGG	Rapid	16-1508	£ 2.05	£ 8.10	
Mini DIN socket	6 way board mount	1 off	Protech LNB	Rapid	20-0690	£ 0.49	£ 0.49	
IC DIL socket 14 pin	0.3” turned pin	1 off	No source	Rapid	22-1721	£ 0.24	£ 0.24	
IC DIL socket 16 pin	0.3” turned pin	1 off	No source	Rapid	22-1722	£ 0.27	£ 0.27	
IDC cable socket	14 way	2 off	Harting 0918 514 7813	Farnell	109-7024	£ 0.91	£ 1.82	
IDC header / pcb plug	14 way R/A	1 off	Harting 0918 514 7913	Farnell	110-6746	£ 0.89	£ 0.89	**SUB**
IDC cable socket	6 way	1 off	Harting 0918 506 7813	Farnell	109-7021	£ 0.60	£ 3.60	**TOTALS**
IDC header / pcb plug	6 way R/A	1 off	Harting 0918 506 7913	Farnell	110-6744	£ 0.67	£ 4.02	**£ 22.23**
Printed circuit board		1 off	GCI_HTSSCAN3b_16AUG07.PCB	PCB pool		£ 18.00	£ 18.00	**£ 50.00**
Electronics board	-------------------	1 off	SSCAN3.SCH	GCI/ROB	---------------------	£ 50.00	£ 50.00	**£ 20.00**

Item	**Description**	**Qty**	**Manufacturer / part #**	**Supplier**	**Part #**	**£ each**	**£ total**	**Note**
Miscellaneous								
Interconnecting cables								
Timing signals to B&H card	15 way High density plug	1 off	McMurdo 15 way D-type	Farnell	107-1808	£ 2.14	£ 2.14	
	15 way High density socket	1 off	McMurdo 15 way D-type	Farnell	107-1811	£ 2.24	£ 2.24	**SUB**
	15 W HD shell	2 off	McMurdo	Farnell	107-5182	£ 1.47	£ 2.94	**TOTALS**
	Braided sleeving 8 mm	1 off	GREMCO PETBK8B10 10 m	Farnell	129-7212	£ 0.54/ m	£ 1.08	**£ 8.40**
Cable construction			SSCAN3.SCH	GCI/ROB			£ 50.00	**£ 50.00**

Item	**Description**	**Qty**	**Manufacturer / part #**	**Supplier**	**Part #**	**£ each**	**£ total**	**Note**
Scanner drive amplifiers								
20 HP panel (1/2 42HP panel)	Front panel	1 off	RS / Schroff 20838146 cut in two	RS	437-1722	£ 7.25	£ 7.25	
Interconnecting cable assembly	--------------------------	2 off	--------------------------------	GCI/ROB	---------------------	£ 10.00	£ 10.00	
Baseplate and shield plate assembly	--------------------------	1 off		GCI/ROB	---------------------	£ 30	£ 30	
Galvanometer	Galvo	2 off	VM1000C	GSI / Lumonics	011-3040105	£ 448.42	£ 896.84	**SUB**
Scanner mirror	9.5 mm Be Ag	2 off	Could go for Si/Ag	GSI / Lumonics	710-767933	£ 292.63	£ 585.26	**TOTALS**
Scanner driver	HS tune, bracket	2 off	MiniSax $642	GSI / Lumonics	002-3005048	£ 337.90	£ 675.8	**£ 7.25**
Scanner driver notch filter	HS tuning	2 off	Part of MiniSaax	GSI / Lumonics	711-7330839	£ 28.40	£ 56.80	**£ 10.00**
Galvo-scanner cable set	2 metre	2 off	Specify length, longer better/ special	GSI / Lumonics	712-764163	£ 34.74	£ 69.48	**£ 70**

Electronic parts	**£588**
Electronics construction	**£620**
Mechanical items	**£85**
Galvo system-specific components	**~£820 (scanner driver) + ~£1482 (galvanometer + mirror) = ~£2300**
Total	**~ £1300 + ~ £2300**

6. Supplier details

GSI Lumonics
Orchard House, Broad Lane, Sykehouse
Goole, E. Yorks, DN14 9AS
Tel: 01405 785-028
Fax: 0049 89 317 07250
Email: nstanley@gsig.com
Website: http://www.gs-scanners.com

Edmund Optics
Tudor House, Lysander Close,
York YO30 4XB,
Tel: 01904 691469,
Fax: 01904 691569
Website: http://www.edmundoptics.com/UK/

Farnell in One
Canal Road, Leeds, LS12 2TU
Tel: 08701 200 200
Fax: 08701 200 201
e-mail: sales@farnell.co.uk
Website: http://uk.farnell.com/

Future Technology Devices Int'l Limited
373 Scotland Street
Glasgow, G5 8QB
Tel: 0141 429 2777
Fax: 0141 429 2758
e-Mail (Sales): sales1@ftdichip.com
Website: http://www.ftdichip.com

Rapid Electronics Ltd
Severalls Lane, Colchester,
Essex CO4 5JS.
Tel: 01206 751166
Website: http://www.rapidonline.com

RS Components Ltd
POBox 99, Corby, Northants, NN17-9RS
Tel: 08457 201201
Fax: 01536-201-501; 405-678
Website:http://rswww.com/

Component manufacturers
Analog Devices
http://www.analog.com/
Cinch
http://www.cinch.com/
Fairchild
http://www.fairchildsemi.com/
Harting
http://www.harting.com/
LCR
http://www.lcr-inc.com/
Linear Technology
www.linear.com
Maxim Integrated Products, Inc.
www.maxim-ic.com
McMurdo
http://www.mcmurdo.uk.com/
Microchip
http://www.microchip.com/
Molex
http://www.molex.com/
National Semiconductor
http://www.national.com/

Component manufacturers
Papst
http://www.papst.de/
Philips
http://www.nxp.com/
Schroff
http://www.schroff.co.uk/
ST Microelectronics
http://www.st.com/
Texas Instruments
http://www.ti.com/
Thomas & Betts
http://www.tnb.com/
Tyco / Greenpar
http://catalog.tycoelectronics.com
Vishay
http://www.vishay.com/
Wima
http://www.wima.com/
Zetex
http://www.zetex.com

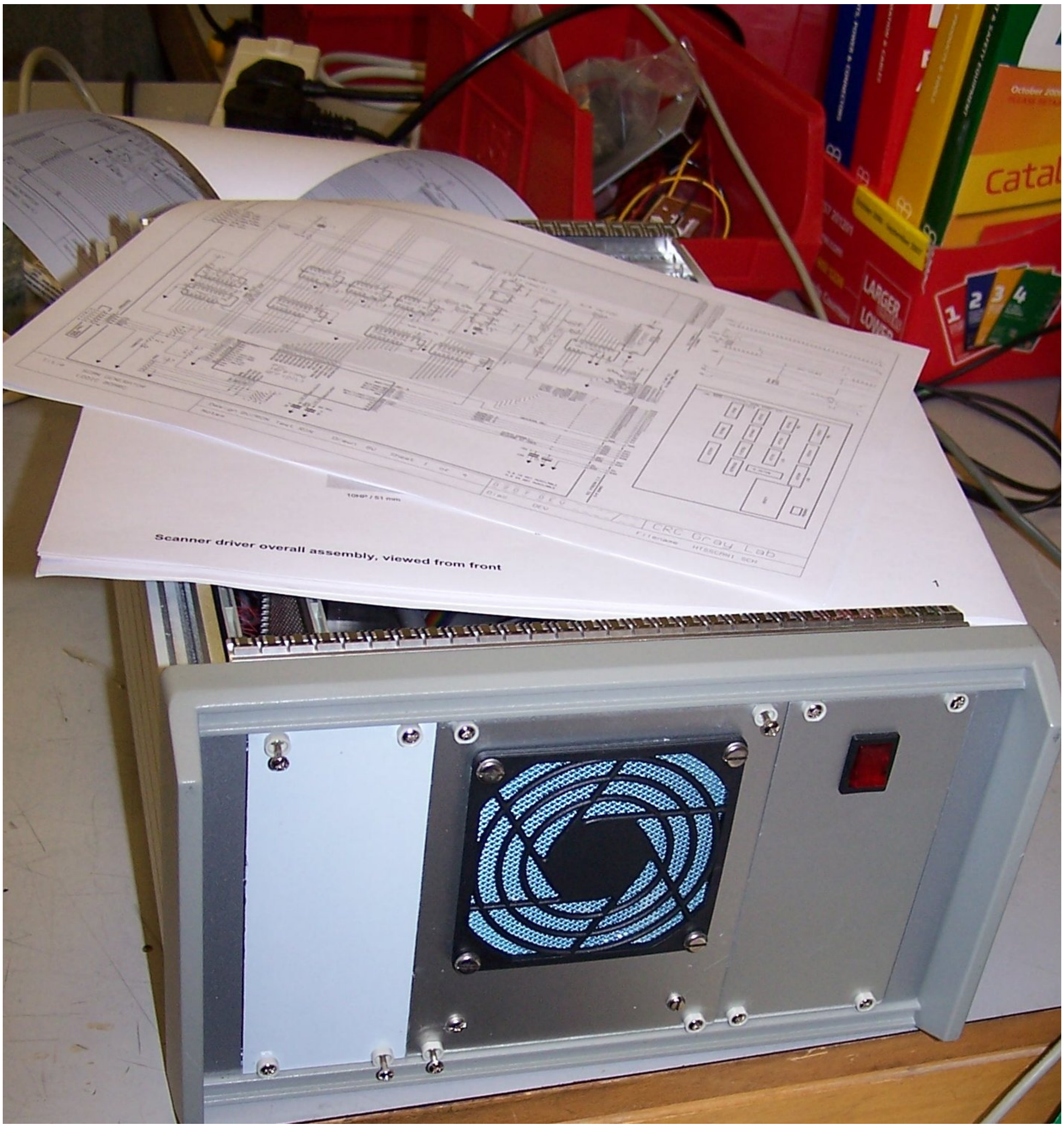


Figure 13. The scanner driver during construction. Yes, we did use the same circuits as shown earlier!

7. PIC firmware and software

As with our other units, we present here listings of the PIC firmware and the test software, along with outline details of how the code is integrated within larger applications. The software used to drive the system is written in two sections: firmware running on the PIC microcontroller and high level host computer software. The firmware was written using a CCS C-code compiler (http://www.ccsinfo.com/) which makes generating the code considerably easier than using an assembler code language such as Microchip MPLAB. The sample code below may be found useful should future modifications be required. In many instances, the high level C-code is developed using National Instruments' LabWindows environment; high level code examples are provided later.

```
PIC code:

/////////////////////////////////////////////////////////////////////////////////////////////////
///  This program uses a PIC16F877A as a slave device on a I2C bus.
///  The address is set under NODE_ADDR define
///  This program uses I2C commands to set digital ports and PWM
///  Controlled using CVI program CVI\Programs\Scan gen\Scan_gen.c
///
///
///   mode0=Sets line bus
///   mode1=Start/stop
///   mode2=Sets frequency and duty cycle of PWM1 output
///   mode3=Sets resolution
///   mode4=Set zoom
///   mode5=Set line offset
///   mode6=Set frame offset
///   mode7=Start/stop frame scan (INT_EXT interrupts)
///   mode8=Sets number of frames
///   mode9=Enable/disable scanner servos
///   mode10=Reverse scanner servo inputs
///   mode11=Stores data to EEPROM
///   mode12=Sets dwell/delay time after end of scan
///
///  17/02/09 version 2 updated SSP interrupt for PCWH 3.249 compiler
///
/////////////////////////////////////////////////////////////////////////////////////////////////

#include "C:\Program Files\PICC\Programs\Scan Generator\Scan Generator_v2\Scan_generator.h"

#define RX_BUF_LEN  10                                                          //This must be 10 or less or it takes too long to clear
#define NODE_ADDR   0x60                                                        //Address used on Kings scanner system for PIC I2C

#byte PIC_SSPSTAT=0x94     // 16f87X bytes

#use i2c(Slave,Slow,sda=PIN_C4,scl=PIN_C3,address=NODE_ADDR,FORCE_HW)           //Don't restart WDT inside SSP interrupt

#use fast_io(A)
#use fast_io(B)
#use fast_io(D)
#use fast_io(E)

//unsigned int slave_buffer[RX_BUF_LEN];
BYTE slave_buffer[RX_BUF_LEN];
BYTE state;

int1 enable_ext_int=0,ext_int_fg=0,StartStopfg=0,done,extStopStartfg=1,scanner_enable=1;  //Flags
int1 reverse_scan_inputs=0,scan_errorfg=1,frameScanfg=0,EnableExtfg=0;

int buffer_index;
int mode;
int period,lsb_duty_cycle,msb_duty_cycle,t2divider=0;
int msb_line_data,lsb_line_data,msb_frame_data,lsb_frame_data;
int resolution,lineStep,zoom,frames=0,frames_temp=0;
int msb_line_offset,lsb_line_offset,msb_frame_offset,lsb_frame_offset;
int dwell_data;

int16 duty_cycle,linesSet,linesSet_temp;

unsigned char read_i2c(void);
void i2c_interrupt_handler(void);
void timer1_interrupt_handler(void);
void i2c_initialize(void);
void i2c_error(void);
void write_i2c(unsigned char transmit_byte);
void SetOutputs(void);
void ReadInput(void);
void InitPWM(void);
void pwm(void);
void DataLatch(int);
void readEEPROM(void);
void stopPWM(void);
void SetFrameScan(void);

#INT_SSP
SSP_isr()
{
    i2c_interrupt_handler();                                          //interrupt happens on every byte received or sent

    if( state==5 )                                                    //data input has read 6 bytes address discarded
    {
      mode=slave_buffer[0];
    switch(mode)    //set the mode of the pic
         {
```

```
  case 0:                                                    //Set 12-bit line
   msb_line_data=slave_buffer[1];
   lsb_line_data=slave_buffer[2];
  break;

  case 1:                                                    //Start/stop
   StartStopfg=slave_buffer[1];                              //Sets flag
if(StartStopfg==1){                                          //If start
   linesSet_temp=linesSet=2048;                              //Reset number of lines to2048
   enable_interrupts(INT_EXT);                               //Enable external interrupt
   output_bit(pin_B1,slave_buffer[1]);                       //Sets start stop line
   pwm();                                                    //Start PWM output

   EnableExtfg=1;                                            //Software scanning started
  }
if(StartStopfg==0 && linesSet_temp==linesSet ){              //If stop and not scanning
   output_bit(pin_B1,0);                                     //Clears start stop line
   disable_interrupts(INT_EXT);                              //Disable external interrupt
   linesSet_temp=2048;                                       //Set frame position to max
   msb_frame_data = linesSet_temp>>8;                        //Split into two bytes
   lsb_frame_data = linesSet_temp & 0xff;
   output_A(msb_frame_data);
   output_D(lsb_frame_data);
   EnableExtfg=0;                                            //Software scanning not started
    }
if(StartStopfg==0 && frameScanfg==1){                        //If stop and line scan
   output_bit(pin_B1,0);                                     //Clears start stop line
   disable_interrupts(INT_EXT);                              //Disable external interrupt
   linesSet_temp=2048;                                       //Set frame position to max
   msb_frame_data = linesSet_temp>>8;                        //Split into two bytes
   lsb_frame_data = linesSet_temp & 0xff;
   output_A(msb_frame_data);
   output_D(lsb_frame_data);
   EnableExtfg=0;                                            //Software scanning not started
    }
  break;

  case 2:                                                    //PWM settings
   period=slave_buffer[1];
   msb_duty_cycle=(slave_buffer[2]);
   duty_cycle=(slave_buffer[2]<<8);
   lsb_duty_cycle=slave_buffer[3];
   duty_cycle=duty_cycle | lsb_duty_cycle;
   t2divider= slave_buffer[4];
  break;

  case 3:                                                    //Sets resolution
   resolution=slave_buffer[1];
  break;

  case 4:                                                    //Set zoom value
   zoom=slave_buffer[1];
  break;

  case 5:                                                    //Set line offset
   msb_line_offset=slave_buffer[1];
   lsb_line_offset=slave_buffer[2];
  break;
  case 6:                                                    //Set frame offset
    msb_frame_offset=slave_buffer[1];
    lsb_frame_offset=slave_buffer[2];
  break;
  case 7:                                                    //Start/stop frame scan
      frameScanfg=slave_buffer[1];                           //Flag to say if frame or line scan 0=frame scan 1=linescan
   if(slave_buffer[1]){
      disable_interrupts(INT_EXT);                           //Disable external interrupt if line scan
      linesSet_temp=1024;                                    //Set frame position to middle
      msb_frame_data = linesSet_temp>>8;                     //Split into two bytes
      lsb_frame_data = linesSet_temp & 0xff;
      output_A(msb_frame_data);
      output_D(lsb_frame_data);
      DataLatch(2);     //Latch line DAC data
   }
   else{
      enable_interrupts(INT_EXT);                            //Enable external interrupt if frame scanning
      linesSet_temp=2048;                                    //Set frame position to max
      msb_frame_data = linesSet_temp>>8;                     //Split into two bytes
      lsb_frame_data = linesSet_temp & 0xff;
   }
  break;
  case 8:                                                    //Sets number of frames
   frames=slave_buffer[1];
   frames_temp=0;
  break;
  case 9:                                                    //Set scanner servo enable line
   scanner_enable=slave_buffer[1];
   output_bit(pin_C0,scanner_enable);
  break;
  case 10:                                                   //Reverse scanner servo inputs
   reverse_scan_inputs=slave_buffer[1];
   output_bit(pin_C1,reverse_scan_inputs);
  break;
  case 11:                                                   //Save data to EEPROM
    write_eeprom (0, period);
    write_eeprom (1,lsb_duty_cycle);
    write_eeprom (2,msb_duty_cycle);
    write_eeprom (3,t2divider);
    write_eeprom (4,resolution);
    write_eeprom (5, reverse_scan_inputs);
    write_eeprom (6, dwell_data);
    write_eeprom (7,frameScanfg);
  break;
  case 12:                                                   //Sets dwell time after end of scan
   dwell_data=slave_buffer[1];
   output_D(dwell_data);                                     //Load port D
```

```
            DataLatch(7);                                             //Latch
           break;
         }

       SetOutputs();

    }

}
#int_EXT
EXT_isr()
{

   if(frames==0){                                                     //Continuous scanning
         output_A(msb_frame_data);
         output_D(lsb_frame_data);

         if(linesSet_temp==2048)
           {
             output_bit(pin_B4,0);                                    //Set frame sync low
             output_bit(pin_B4,1);                                    //Set frame sync high
           }
          DataLatch(2);                                               //Latch line DAC data

         if(linesSet_temp==0)
           {
               if(StartStopfg==0){                                    //If stop
                  output_bit(pin_B1,0);                               //Sets start stop line
                  disable_interrupts(INT_EXT);                        //Disable external interrupt
                  linesSet_temp=2048;                                 //Set frame position to max
                  msb_frame_data = linesSet_temp>>8;                  //Split into two bytes
                  lsb_frame_data = linesSet_temp & 0xff;
                  output_A(msb_frame_data);
                  output_D(lsb_frame_data);
                  DataLatch(2);                                       //Latch line DAC data
                  EnableExtfg=0;                                      //Software scanning stopped
                  output_bit(pin_B4,0);                               //Set frame sync low
                  output_bit(pin_B4,1);                               //Set frame sync high
                  stopPWM();
                 }
               linesSet_temp=linesSet;                                //Reset line count
            }
            else
            {
         linesSet_temp = linesSet_temp-lineStep;                      //Decrease the count
         msb_frame_data = linesSet_temp>>8;                           //Split into two bytes
         lsb_frame_data = linesSet_temp & 0xff;
            }
   }
      else{                                                           //Counting frames
            output_A(msb_frame_data);
            output_D(lsb_frame_data);

         if(linesSet_temp==2048)
           {
             output_bit(pin_B4,0);                                    //Set frame sync low
             output_bit(pin_B4,1);                                    //Set frame sync high
           }
          DataLatch(2);                                               //Latch line DAC data

         if(linesSet_temp==0)
         {
             if(StartStopfg==0){                                      //If stop
                  output_bit(pin_B1,0);                               //Sets start stop line
                  disable_interrupts(INT_EXT);                        //Disable external interrupt
                  linesSet_temp=2048;                                 //Set frame position to max
                  msb_frame_data = linesSet_temp>>8;                  //Split into two bytes
                  lsb_frame_data = linesSet_temp & 0xff;
                  output_A(msb_frame_data);
                  output_D(lsb_frame_data);
                  DataLatch(2);                                       //Latch line DAC data
                  EnableExtfg=0;                                      //Software scanning stopped
                  output_bit(pin_B4,0);                               //Set frame sync low
                  output_bit(pin_B4,1);                               //Set frame sync high
                  stopPWM();
                }

             linesSet_temp=linesSet;                                  //Reset line count

              frames_temp++;
             if(frames_temp==frames){                                 //Stop scanning
                   output_bit(pin_B1,0);                              //Sets start stop line
                   disable_interrupts(INT_EXT);                       //Disable external interrupt
                   linesSet_temp=2048;                                //Set frame position to max
                   msb_frame_data = linesSet_temp>>8;                 //Split into two bytes
                  lsb_frame_data = linesSet_temp & 0xff;
                  output_A(msb_frame_data);
                  output_D(lsb_frame_data);
                  DataLatch(2);                                       //Latch line DAC data
                  output_bit(pin_B4,0);                               //Set frame sync low
                  output_bit(pin_B4,1);                               //Set frame sync high
                  stopPWM();
               }
           }
           else
           {
         linesSet_temp = linesSet_temp-lineStep;                      //Decrease the count
         msb_frame_data = linesSet_temp>>8;                           //Split into two bytes
         lsb_frame_data = linesSet_temp & 0xff;
           }
      }

}
//-------------------------------------------------------------------------------
void i2c_interrupt_handler(void)
```

```
{
BYTE incoming;
unsigned char  tx_byte;

          state = i2c_isr_state();

          if(state < 0x80)                                            //Master is sending data
          {
                    incoming = i2c_read();                            //State 0 is address

          switch(state){
      case 1:                                                         //First data byte
                              slave_buffer[(state-1)] = incoming;
         break;
          case  2:                                                    //Second data
                              slave_buffer[(state-1)] = incoming;
           break;
      case  3:                                                        //Third data
                              slave_buffer[(state-1)] = incoming;
           break;
      case  4:                                                        //Fourth
                              slave_buffer[(state-1)] = incoming;
           break;
      case  5:                                                        //Fifth
                              slave_buffer[(state-1)] = incoming;
           break;
      case  6:                                                        //Sixth
                              slave_buffer[(state-1)] = incoming;
           break;
      case  7:                                                        //Seventh
                              slave_buffer[(state-1)] = incoming;
           break;
      }
          }

if(state == 0x80 )                                                    //Master is requesting data
          {
      buffer_index = 0;                                               // Reset the buffer index
      ReadInput();                                                    // Read bytes into buffer
      buffer_index = 0;                                               // Reset the buffer index
      tx_byte = slave_buffer[buffer_index];                           // Get byte from the buffer
    //  i2c_write(tx_byte);   //Does not work to well with long I2C connections

      #asm                                                            //Assembler for write command
         MOVF   tx_byte,W
         MOVWF  0x66
         MOVF   0x13,W
         MOVF   0x66,W
         MOVWF  0x13                                                  //Set data into buffer
         nop                                                          //Delay before releasing clock
         nop                                                          //10 nop ie 2us
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         BSF    0x14.4                                                //Release clock
         BCF    0x0C.3                                                //Clear SSP interrupt flag

TEST_BF:
         BSF    0x03.5                                                //Change to bank 1
         BTFSS  0x14.0                                                //Test BF bit
         GOTO   BF_OK
         BCF    0x03.5                                                //Change to bank 0
         GOTO   TEST_BF

BF_OK:
         CLRF   0x78
         BCF    0x03.5                                                //Change to bank 0
  #endasm

      buffer_index++;                                                 // increment the buffer index

      break;
          }
      if(state > 0x80 ){
         tx_byte = slave_buffer[buffer_index];                        // Get byte from the buffer
         #asm                                                         //Assembler for write command
         MOVF   tx_byte,W
         MOVWF  0x66
         MOVF   0x13,W
         MOVF   0x66,W
         MOVWF  0x13                                                  //Set data into buffer
         nop                                                          //Delay before releasing clock
         nop                                                          //10 nop ie 2us
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         nop
         BSF    0x14.4                                                //Release clock
         BCF    0x0C.3                                                //Clear SSP interrupt flag

TEST_BF_1:
         BSF    0x03.5                                                //Change to bank 1
         BTFSS  0x14.0                                                //Test BF bit
         GOTO   BF_OK_1
         BCF    0x03.5                                                //Change to bank 0
         GOTO   TEST_BF_1
BF_OK_1:
         CLRF   0x78
         BCF    0x03.5                                                //Change to bank 0
  #endasm

         buffer_index++;                                              //Increment the buffer index

         break;
      }

}
//------------------------------------------------------------------------------
void SetOutputs()
{
switch(mode)                                                          //set the mode of the pic
   {
      case 0:                                                         //Set 12-bit line
      output_A(msb_line_data);
      output_D(lsb_line_data);
      DataLatch(0);                                                   //Latch line data
      break;
      case 1:                                                         //Not used
      break;
      case 2:                                                         //Not used
      break;
      case 3:                                                         //Set resolution
        switch(resolution)
        {
        case 1:                                                       //2048x2048
         output_D(1);
         DataLatch(1);                                                //Latch pixel clock divider latch
         lineStep=1;
         break;
         case 2:                                                      //1024x1024
         output_D(1);
         DataLatch(1);                                                //Latch pixel clock divider latch
         lineStep=2;
         break;
         case 3:                                                      //512x512
         output_D(3);
         DataLatch(1);                                                //Latch pixel clock divider latch
         lineStep=4;
         break;
         case 4:                                                      //256x256
         output_D(7);
         DataLatch(1);                                                //Latch pixel clock divider latch
         lineStep=8;
         break;
         case 5:                                                      //128x128
         output_D(15);
         DataLatch(1);                                                //Latch pixel clock divider latch
           lineStep=16;
         break;
         case 6:
         output_D(31);                                                //64x64
         DataLatch(1);                                                //Latch pixel clock divider latch
         lineStep=32;
         break;
         case 7:
         output_D(63);                                                //32x32
         DataLatch(1);                                                //Latch pixel clock divider latch
         lineStep=64;
         break;
        }
        linesSet_temp=linesSet=2048;                                  //Reset number of lines to2048
        msb_frame_data = linesSet_temp>>8;                            //Split into two bytes
        lsb_frame_data = linesSet_temp & 0xff;
      break;
      case 4:                                                         //Set zoom
         output_D(zoom);
         DataLatch(3);                                                //Latch Y zoom
         DataLatch(4);                                                //Latch X zoom
      break;
      case 5:                                                         //Set line offset
         output_A(msb_line_offset);
         output_D(lsb_line_offset);
         DataLatch(5);                                                //Latch line offset
      break;
      case 6:                                                         //Set frame offset
         output_A(msb_frame_offset);
         output_D(lsb_frame_offset);
         DataLatch(6);                                                //Latch frame offset
      break;
   }
}
//-----------------------------------------------------------------------
void ReadInput()
{
   switch(mode)
   {
      case 254:
      slave_buffer[buffer_index] = (scan_errorfg & 0x01);             // Put scanner servo error flag into buffer
      break;
   }
}
//-----------------------------------------------------------------------
 void DataLatch(output_latch)
 {
          output_E(output_latch);                                     //Select output
          Delay_us(1);
          output_bit(pin_B2,1);
          output_bit(pin_B2,0);
 }
//-----------------------------------------------------------------------
```

```
void pwm()
{
          set_pwm1_duty(duty_cycle);                                  //Set timer2 duty cycle in PWM mode
          setup_timer_2(t2divider,period,16);                         //setup_timer_2(mode,period(0-255),postscale)
                                                                      //and enable counter
}
//-----------------------------------------------------------------------
void InitPWM()                                                        //Sets the PIC running at 100ns at 50Hz
{
      period=9;                                                       //Set the PWM state to 500KHz 50ns width
      duty_cycle=1;
      t2divider= 4;
      setup_ccp1(CCP_PWM);                                            //Setup CCP to be PWM output
      set_pwm1_duty(duty_cycle);                                      //Set timer2 duty cycle in PWM mode
      setup_timer_2(t2divider,period,16);                             //setup_timer_2(mode,period(0-255),postscale) and
                                                                      //enable counter

}
//-----------------------------------------------------------------------
void stopPWM()
{
      setup_timer_2(T2_DISABLED,period,16);                           //Disable PWM output
}

//------------------------------------------------------------------------
ExtStartStop()                                                        //External start stop signal detect on B5
{
int1 state;

      state=input_state(pin_B5);
      if(state==0 && extStopStartfg==0){                              //Start scanning
        StartStopfg=1;                                                //Set flag
        linesSet_temp=linesSet=2048;                                  //Reset number of lines to 2048
        frames=0;                                                     //Continuous scanning
        readEEPROM();                                                 //Read EEPROM data
        pwm();                                                        //Set PWM
        mode=3;                                                       //Set resolution
        SetOutputs();
        output_bit(pin_C1,reverse_scan_inputs);                       //Set scanner servo reversal pin
        output_bit(pin_C0,0);                                         //Enable scanner servos
        output_bit(pin_B1,1);                                         //Sets start/stop line B1 high
        extStopStartfg=1;                                             //Set flag
        output_D(dwell_data);                                         //Load dwell time on port D
        DataLatch(7);                                                 //Latch data
        SetFrameScan();
      }
     if(state==1 && extStopStartfg==1){                               //Stop scanning
        StartStopfg=0;                                                //Clear flag
         output_bit(pin_B1,0);                                        //Sets start stop line B1 low
         extStopStartfg=0;
         disable_interrupts(INT_EXT);                                 //Disable external interrupt
         linesSet_temp=2048;                                          //Set frame position to max
         msb_frame_data = linesSet_temp>>8;                           //Split into two bytes
         lsb_frame_data = linesSet_temp & 0xff;
         output_A(msb_frame_data);
         output_D(lsb_frame_data);
         DataLatch(2);                                                //Latch line DAC data
         output_A(0x08);                                              //Set line offset
         output_D(0x00);
         DataLatch(5);                                                //Latch line offset
         output_A(0x08);                                              //Set frame offset
         output_D(0x00);
         DataLatch(6);                                                //Latch frame offset
      }
}
//-----------------------------------------------------------------------
void readEEPROM(void)                                                 //Read data from EEPROM
{
         period=read_eeprom (0);
         lsb_duty_cycle=read_eeprom (1);
         msb_duty_cycle=read_eeprom (2);
         duty_cycle=(msb_duty_cycle<<8 | lsb_duty_cycle);
         t2divider=read_eeprom (3);
         resolution=read_eeprom (4);
         reverse_scan_inputs=read_eeprom (5);
         dwell_data=read_eeprom (6);
         frameScanfg=read_eeprom (7);
}
//-------------------------------------------------------------------------
void SetFrameScan(void)                                               //Set to line or frame scan on external on/off
{
      if(frameScanfg){
         disable_interrupts(INT_EXT);                                 //Disable external interrupt
         linesSet_temp=1024;                                          //Set frame position to middle
         }
            else{
               enable_interrupts(INT_EXT);                            //Enable external interrupt
               linesSet_temp=2048;                                    //Set frame position to max
            }
}
//--------------------------------------------------------------------------
void i2c_initialize(void)
{
   port_b_pullups(TRUE);

   set_tris_A(0x20);                                                  //Set ports to all outputs except A5
   set_tris_B(0x21);                                                  //Set all to outputs except B0 and B5 to input
   set_tris_D(0x00);
   set_tris_E(0x00);
   output_bit(pin_B2,0);                                              //Set latch pin low
   output_bit(pin_B4,1);                                              //Set frame sync high
   output_bit(pin_C0,1);                                              //Set scanner enable high
   output_bit(pin_C1,1);                                              //Set scanner inputs to normal
   setup_timer_0(RTCC_INTERNAL);setup_wdt(WDT_144MS);
   linesSet_temp=linesSet=2048;                                       //Initalise number of lines to 2048
   msb_frame_data = linesSet_temp>>8;                                 //Split into two bytes
   lsb_frame_data = linesSet_temp & 0xff;
   setup_ccp1(CCP_PWM);
   InitPWM();                                                         //Set the PIC initally running in PWM

   output_A(0x08);                                                    //Set msb on 12-bit line bus
   output_D(0x00);
   DataLatch(0);                                                      //Latch line data

   output_D(1);                                                       //Resolution 1024x1024
   DataLatch(1);                                                      //Latch pixel clock divider latch
   linesSet_temp=linesSet=2048;                                       //Reset number of lines to 2048
   lineStep=2;

   output_D(255);                                                     //Set zoom
   DataLatch(3);                                                      //Latch Y zoom
   DataLatch(4);                                                      //Latch X zoom

   output_A(0x08);                                                    //Set line offset
   output_D(0x00);
   DataLatch(5);                                                      //Latch line offset

   output_A(0x08);                                                    //Set frame offset
   output_D(0x00);
   DataLatch(6);                                                      //Latch frame offset

   enable_interrupts(INT_SSP);                                        //Enable MSSP interrupts
   disable_interrupts(INT_EXT);                                       //Disable external interrupt
   ext_int_edge( l_TO_h );                                            //High to low interrupt
}
void main() {

   i2c_initialize();
   enable_interrupts(GLOBAL);
   PIC_SSPSTAT = 0x00;                                                //Clear the SSPSTAT register
   while (1)
    {
          restart_wdt();                                              //Restart watchdog timer

      if(EnableExtfg==0 )                                             //Software scanning not started
      {
          ExtStartStop();                                             //Check for external start stop command
       }

       if( input(PIN_A5)==0 )                                         //No error detect on scanner servos
       {
         scan_errorfg=0;                                              //Set error flag
       }
      else{                                                           //Error
            scan_errorfg=1;                                           //Set error flag
       }
    }
}
```

CVI code:

```
//This program controls a PIC 16F877A programmed as a slave I2C device
//using a C compiled program PICC\Scan Generator\Scan_generator.c using the
//CCS compiler PCWH 3.249
//
//19/06/06
//
//

#include "cvixml.h"
#include <rs232.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include "utility.h"
#include "formatio.h"
#include  <analysis.h>
#include "DeviceFinder.h"
#include "Scan_gen.h"
#include "Scan_gen_ui.h"
#include "IO_interface_v2.h"
#include "usbconverter_v2.h"

#define Round  RoundRealToNearestInteger
#define address   0                               //Programable address of SCANNERPIC I/O chip the base address of PIC is 0x60
#define bus 2                                     //Set to required bus(MPTR system) else set to 2
#define Stop        0                             //Start /stop codes
#define Start       1

static int PORT;
static int ScanGenpanel;
static int mode,hyst_offset,line_scan, zoom,err;
static int clock1,startScanfg=0,scanOKfg=1,dwell,spareBits=0;                               //spareBits are the lower 4 //bits on the
                                                                                            dwell time latch
static double framePulses;

int comPort(void);
int setup(void);
int comtest(void);
int SetPWRepRate(void);
int LoadLineData(int);
void StartStop(int);
int SetSpeed(void);
int SetResolution(void);
int SetZoom(void);
int SetX_shift(void);
int SetY_shift(void);
int SetDwellTime(int,int);
int RevScan(void);
int LineScan(void);

int main (int argc, char *argv[])
{
          if (InitCVIRTE (0, argv, 0) == 0)
                    return -1;                                                            /* out of memory */
          if ((ScanGenpanel = LoadPanel (0, "Scan_gen_ui.uir", PANEL)) < 0)
                    return -1;
          DisplayPanel (ScanGenpanel);
          GCI_closeI2C();
          Delay(0.5);
          GCI_Init_USB();
          GCI_setI2Cport(PORT);                                                           //Set port number
          Delay(0.2);                                                                     //Delay for settling
          GCI_InitScanGen();                                                              //Initialise scanner settings
          GCI_EnableLowLevelErrorReporting(1);
          SetCtrlAttribute (ScanGenpanel, PANEL_TIMER, ATTR_ENABLED, 1);                  //Enable timer
          RunUserInterface ();
          StartStop(Stop);                                                                //Stop scanner
          DiscardPanel (ScanGenpanel);
          GCI_closeI2C();
          return 0;
}
int GCI_Init_USB()
{
int numberOfDevices = 0;
int err;
char PID[200];
char *LLPG1;
char *LLPG2;
char *LLPG3;
char *curfname1;
char *curfname2;
char *curfname3;
char *curfname4;

          LLPG1="VID_0403+PID_6001+DPC2BV54A";                                  //VID, PID and serial number codes of FTDI245 devices
          LLPG2="VID_0403+PID_6001+DPC1ICWGb";
          LLPG3="VID_0403+PID_6001+DPC1ICWGc";

          curfname1="LLPG1.txt";
          curfname2="LLPG2.txt";
          curfname3="LLPG3.txt";
          curfname4="LLPGdefault.txt";

          getDevices(PID);                                                      //Returns PID of FTDI245

          numberOfDevices = getNumberOfDevices();
          if(numberOfDevices == 0)
          {
                    MessagePopup ("Connection problem", "Please ensure the USB cable is connected and try again");
                    getDevices(PID);                                            //Returns PID of FTDI245
                    numberOfDevices = getNumberOfDevices();
          }


            if(numberOfDevices >1)
          {
                    MessagePopup ("Connection problem", "More than one device connected");
                    return -1;
          }
          if(numberOfDevices == 1)
          {
                    PORT = getPorts()[0];

                    comPort();
          }
                    else{
                              MessagePopup ("Connection problem", "USB communications problem");
                               return -1;
                              }
/*              if(strcmp (PID,LLPG1 )==0){ RestoreCalData(curfname1);
                              SetCtrlVal (calPanel, CAL_PANEL_GEN_ID2 , "LLPG1");  }
                              else if(strcmp (PID,LLPG2 )==0){ RestoreCalData(curfname2);
                                        SetCtrlVal (calPanel, CAL_PANEL_GEN_ID2 , "LLPG2");}
                                        else if(strcmp (PID,LLPG3 )==0){ RestoreCalData(curfname3);
                                                  SetCtrlVal (calPanel, CAL_PANEL_GEN_ID2 , "LLPG3");}
                                                  else { RestoreCalData(curfname4);
                                                            SetCtrlVal (calPanel, CAL_PANEL_GEN_ID2 , "?");}
                      */
                    comtest();
             return 0;
}
int comPort()
{
          if(setup() != 0)
          {
                    MessagePopup ("Port Problem","Port cannot be used");
                    return -1;
          }
          return 0;                                                             //when port is OK, code will run to here
}
int setup()
{
int status;

          status = OpenComConfig (PORT, "", 9600, 0, 8, 1, 164, 164);
          if(status < 0)
          {
          return -1;
          }
          SetComTime (PORT, 1.0);                           //Set port time-out
          FlushInQ (PORT);                                  //Flush port
          FlushOutQ (PORT);
          return 0;
}
int comtest(void)                                           //Test I2C communications
{
            err=SetZoom();
            if(err!=0){
                     MessagePopup ("I2C error", "Is the unit switched on?");
                    }

            return 0;
}
void GCI_InitScanGen(void)                                  //Initialise settings and PIC variables
{
int hyst_offset;

SetSpeed();                                                 //Sets speed and also sets hysteresis offset
SetResolution();                                            //Set resolution
StartStop(Stop);                                            //Make sure scanning has stopped
SetZoom();                                                  //Set zoom
SetX_shift();
SetY_shift();
RevScan();                                                  //Set scanning direction
LineScan();                                                 //Enable line/frame scan

}

int LoadLineData(int line)
{
char val1[20];
int msb_line,lsb_line,msb_frame,lsb_frame;

msb_line = line >>8;
lsb_line = line & 0xff;
mode=0;                                                     //Load line data mode

          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=(msb_line | 0x08);                        //Keep msb high
          val1[3]=lsb_line;
          GCI_writeI2C(6, val1, bus);

          return 0;

}
int SetRepRate(int rep_val,int rep_div_1,int rep_div_2)
{
char val1[20];
double frequency,double_duty_cycle,duty_cycle_freq;
int divider,duty_cycle;
int msb_duty_cycle,lsb_duty_cycle;

          switch(rep_div_1)                                 //Calculate the frequency
          {
                    case 0:
                    frequency=0;
                    divider=1;
                    break;
                    case 4:
                    divider=1;
                    frequency=20E6/(((rep_val+1)*divider*4)*1000);
                    break;
                    case 5:
                    divider=4;
                    frequency=20E6/(((rep_val+1)*divider*4)*1000);
                    break;
                    case 6:
                    divider=16;
                    frequency=20E6/(((rep_val+1)*divider*4)*1000);
                    break;
                    default:
                    divider=1;
          }

          //Calculate duty cycle bytes
          double_duty_cycle = 0.005*frequency;   //Fixes pulse width to 50 ns
          duty_cycle_freq=frequency*(100.0/double_duty_cycle);
          duty_cycle=20E3/( duty_cycle_freq*divider);
          msb_duty_cycle=duty_cycle>>8;
          lsb_duty_cycle=duty_cycle & 0xff;

          mode=2;                                           //Set PWM mode
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;           //Set PWM mode
          val1[2]=rep_val;
          val1[3]=msb_duty_cycle;
          val1[4]=lsb_duty_cycle;
          val1[5]=rep_div_1;
          GCI_writeI2C(6, val1, bus);                       //Write I2C

          return 0;
}
int SetSpeed()
{
double set_frequency,set_divider;
int speed;

                    GetCtrlVal(ScanGenpanel, PANEL_SPEED,&speed);
                    GetCtrlVal(ScanGenpanel, PANEL_HYST_OFFSET,&hyst_offset);
                    if(hyst_offset>=2047){
                              hyst_offset=2047;
                              SetCtrlVal(ScanGenpanel, PANEL_HYST_OFFSET,2047);
                    }
switch(speed){
case 1:                                                     //Very fast
```

```
set_frequency=800;                                //Set frequency in KHz
set_divider=Round((5000.0/set_frequency)-1);
SetRepRate(set_divider,4,1);
//SetRepRate(6,4,1);                              //714.3 KHz
LoadLineData(hyst_offset);                        //This number is or'ed with 0x80 in the routine
clock1= ceil (set_frequency);
SetDwellTime(0x50,spareBits);
//clock1=715;
break;
case 2:                                           //Fast
set_frequency=400;                                //Set frequency in KHz
set_divider=Round((5000.0/set_frequency)-1);
SetRepRate(set_divider,4,1);
//SetRepRate(12,4,1);                             //384.6 KHz
hyst_offset=Round(hyst_offset/2);
LoadLineData(hyst_offset);
clock1= ceil (set_frequency);
SetDwellTime(0x40,spareBits);
//        clock1=385;
break;
case 3:                                           //Normal
set_frequency=200;                                //Set frequency in KHz
set_divider=Round((5000.0/set_frequency)-1);
SetRepRate(set_divider,4,1);
//        SetRepRate(24,4,1);                     //200 KHz
hyst_offset=Round(hyst_offset/5);
LoadLineData(hyst_offset);
clock1= ceil (set_frequency);
SetDwellTime(0x30,spareBits);
//        clock1=200;
break;
case 4:                                           //Slow
set_frequency=100;                                //Set frequency in KHz
set_divider=Round((5000.0/set_frequency)-1);
SetRepRate(set_divider,4,1);
//        SetRepRate(48,4,1);                     //102 KHz
hyst_offset=Round(hyst_offset/10);
LoadLineData(hyst_offset);
clock1= ceil (set_frequency);
SetDwellTime(0x20,spareBits);
//        clock1=102;
break;
case 5:                                           //Very slow
set_frequency=50;                                 //Set frequency in KHz
set_divider=Round((5000.0/set_frequency)-1);
SetRepRate(set_divider,4,1);
//        SetRepRate(96,4,1);                     //51.5 KHZ
hyst_offset=Round(hyst_offset/50);
LoadLineData(hyst_offset);
clock1= ceil (set_frequency);
SetDwellTime(0x10,spareBits);
//        clock1=52;
break;
case 6:                                           //Slowest
set_frequency=25;                                 //Set frequency in KHz
set_divider=Round((5000.0/set_frequency)-1);
SetRepRate(set_divider,4,1);
//        SetRepRate(192,4,1);                    //25.91 KHz
hyst_offset=Round(hyst_offset/100);
LoadLineData(hyst_offset);
clock1= ceil (set_frequency);
SetDwellTime(0x00,spareBits);
//        clock1=26;
break;
}
return 0;
}
int SetDwellTime(dwell,spareBits)                 //Set dwell/delay after end of scan (upper 4 bits)
{                                                 //Lower 4 bits spare
char val1[20];

          dwell=dwell;
          mode=12;                                //Set to dwell time mode
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=dwell | spareBits;

                    GCI_writeI2C(6, val1, bus);

             return 0;
}
int SetResolution(void)
{
char val1[20];
int resolution;

GetCtrlVal(ScanGenpanel, PANEL_RESOLUTION,&resolution);
          mode=3;                                               //Set resolution
          switch(resolution){
          case 1:                                               //2048x2048
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=1;
          GCI_writeI2C(6, val1, bus);
          framePulses=2048;
          break;
          case 2:                                               //1024x1024
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=2;
          GCI_writeI2C(6, val1, bus);
          framePulses=1024;
          break;
          case 3:                                               //512x512
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=3;
          GCI_writeI2C(6, val1, bus);
          framePulses=512;
          break;
          case 4:                                                     //256x256
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=4;
          GCI_writeI2C(6, val1, bus);
          framePulses=256;
          break;
          case 5:                                                     //128x128
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=5;
          GCI_writeI2C(6, val1, bus);
          framePulses=128;
          break;
          case 6:                                                     //64x64
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=6;
          GCI_writeI2C(6, val1, bus);
          framePulses=64;
          break;
          case 7:                                                     //32x32
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=7;
          GCI_writeI2C(6, val1, bus);
          framePulses=32;
          break;
          }

          return 0;
}
int SetZoom(void)
{
char val1[20];
int err;


GetCtrlVal(ScanGenpanel, PANEL_ZOOM,&zoom);
          mode=4;                                                     //Zoom mode
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=zoom;                                               //Set zoom value x1,x2,x5,x10,x20 or park
          err=GCI_writeI2C(6, val1, bus);

          return err;

}
int SetX_shift(void)
{
char val1[20];
int x_shift,msb_x_shift,lsb_x_shift;

GetCtrlVal(ScanGenpanel, PANEL_X_SHIFT,&x_shift);
          mode=6;                                                     //Line shift mode
          msb_x_shift = x_shift >>8;
          lsb_x_shift = x_shift & 0xff;
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=msb_x_shift;                                        //Set line shift value
          val1[3]=lsb_x_shift;
          GCI_writeI2C(6, val1, bus);

          return 0;
}
int SetY_shift(void)
{
char val1[20];
int y_shift,msb_y_shift,lsb_y_shift;

GetCtrlVal(ScanGenpanel, PANEL_Y_SHIFT,&y_shift);
          mode=5;                        //Frame shift mode
          msb_y_shift = y_shift >>8;
          lsb_y_shift = y_shift & 0xff;
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=msb_y_shift;                                        //Set frame shift value
          val1[3]=lsb_y_shift;
          GCI_writeI2C(6, val1, bus);

          return 0;
}
int RevScan(void)
{
char val1[20];
int val;

GetCtrlVal(ScanGenpanel, PANEL_REV_SCAN ,&val);

          mode=10;                                                    //Reverse scanner inputs
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=val;
          GCI_writeI2C(6, val1, bus);
          return 0;
}
int LineScan(void)                                                    //Enable line/frame scan
{
char val1[20];
```

```
GetCtrlVal(ScanGenpanel, PANEL_LINE_SCAN ,&line_scan);
          mode=7;                        //Enable line/frame scan
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          val1[2]=line_scan;                                          //0=enable frame scan; 1=disable frame scan
          GCI_writeI2C(6, val1, bus);

          return 0;

}
void StartStop(int val)
{
char val1[20];
int frames;
double timer1,timer2,frameTime;

if(val==1){                             //Start, so set number of frames
          SetCtrlAttribute (ScanGenpanel,PANEL_START_SCAN , ATTR_DIMMED,1 );
          SetCtrlAttribute (ScanGenpanel,PANEL_FRAME_NUM , ATTR_DIMMED,1 );
          GetCtrlVal(ScanGenpanel, PANEL_FRAME_NUM ,&frames);
          mode=8;                                                     //Set number of frames to be scanned
          val1[0]=SCANNERPIC | (address <<1);
          val1[1]=mode;
          if(line_scan==0)                                            //Not a line scan
          val1[2]=frames;                                             //Number of frames to be scanned; 0==continuous scanning
          else{
                    val1[2]=0;                                        //Line scan so continuous scanning
                    frames=0;                }
                    GCI_writeI2C(6, val1, bus);
                              }
                              mode=1;                                 //Start/stop mode
                              val1[0]=SCANNERPIC | (address <<1);
                              val1[1]=mode;
                              val1[2]=val;                            //0=stop 1=start
                              GCI_writeI2C(6, val1, bus);

          if(line_scan==1&&val==1){                                   //1==line scan only, so disable frame scan
                              mode=7;                                 //Start/stop frame scan
                              val1[0]=SCANNERPIC | (address <<1);
                              val1[1]=mode;
                              val1[2]=1;                              //1==disable frame scan
                              GCI_writeI2C(6, val1, bus);
                     }
                    SetCtrlVal(ScanGenpanel, PANEL_SCAN_ON_IND ,val);
                    SetCtrlAttribute (ScanGenpanel,PANEL_START_SCAN , ATTR_DIMMED,val);
                    SetCtrlAttribute (ScanGenpanel,PANEL_FRAME_NUM , ATTR_DIMMED,val );
                    ProcessDrawEvents ();
if(val==1&&frames>=1){
                              timer1=Timer();
                              timer2=Timer();
while((timer2-timer1)<=frames*(framePulses/(clock1/(3.0+(hyst_offset/1000)))))
          {
                    ProcessDrawEvents ();
                    timer2=Timer();
                    ProcessSystemEvents ();
          }
                    SetCtrlAttribute (ScanGenpanel,PANEL_START_SCAN , ATTR_DIMMED,0 );
                    SetCtrlAttribute (ScanGenpanel,PANEL_FRAME_NUM , ATTR_DIMMED,0 );
                    SetCtrlVal(ScanGenpanel, PANEL_SCAN_ON_IND ,0);
}

}
int CVICALLBACK cbstart_scan (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];
int val;

          switch (event)
                    {
                    case EVENT_COMMIT:
                    if(zoom!=255){
                                 SetZoom();
                              }
                    SetSpeed();                                       //Set speed
                    SetResolution();                                  //Set resolution
                    RevScan();                                        //Set set scanning direction
                    GetCtrlVal(ScanGenpanel, PANEL_SCAN_ENABLE ,&val);
                    if(val==0){StartStop(Start);
                    startScanfg=1;                                    //Set the flag
                    }
                    break;
                    }
          return 0;
}

int CVICALLBACK cbstop (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];

          switch (event)
                    {
                    case EVENT_COMMIT:
                    StartStop(Stop);                                  //Stop scanning
                    startScanfg=0;                                    //Clear the flag
                    break;
                    }
          return 0;
}

int CVICALLBACK cbzoom (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];
```

```
          switch (event)
                    {
                    case EVENT_COMMIT:
                    GetCtrlVal(ScanGenpanel, PANEL_ZOOM,&zoom);
                    if(startScanfg==1){                                   //Change zoom only when scanning
                              SetZoom();
                    }
                              break;
                    }
          return 0;
}

int CVICALLBACK cby_shift (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];
int y_shift,msb_y_shift,lsb_y_shift;

          switch (event)
                    {
                    case EVENT_COMMIT:
                    SetY_shift();
                    break;
                    }
          return 0;
}

int CVICALLBACK cbspeed (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
int speed;

          switch (event)
                    {
                    case EVENT_COMMIT:
                    SetSpeed();
                    break;
                    }
          return 0;
}

int CVICALLBACK cbresolution (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];
int resolution;

          switch (event)
                    {
                    case EVENT_COMMIT:
                    SetResolution();
                    break;
                    }
          return 0;
}

int CVICALLBACK cbx_shift (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];
int x_shift,msb_x_shift,lsb_x_shift;

          switch (event)
                    {
                    case EVENT_COMMIT:
                    SetX_shift();
                    break;
                    }
          return 0;
}

int CVICALLBACK cbquit (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];

          switch (event)
                    {
                    case EVENT_COMMIT:
                    mode=9;                                                     //Enable/disable scanner
                    val1[0]=SCANNERPIC | (address <<1);
                    val1[1]=mode;
                    val1[2]=1;                                                  //Enable=0 standby=1
                    GCI_writeI2C(6, val1, bus);                                 //Scanners to standby

                    StartStop(Stop);                                            //Stop scanning

                    mode=11;                                                    //Store data into EEPROM
                    val1[0]=SCANNERPIC | (address <<1);
                    val1[1]=mode;
                    GCI_writeI2C(6, val1, bus);
                    Delay(0.2);                                                 //Takes time to write to EEPROM
                    QuitUserInterface (0);
                    break;
                    }
          return 0;
}

int CVICALLBACK cbhyst_offset (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
```

```
                    SetSpeed();
                    break;
                    }
          return 0;
}

int CVICALLBACK cbshift_reset (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
          switch (event)
                    {
                    case EVENT_COMMIT:
                    SetCtrlVal(ScanGenpanel, PANEL_X_SHIFT ,2047);
                    SetCtrlVal(ScanGenpanel, PANEL_Y_SHIFT ,2047);
                    SetX_shift();
                    SetY_shift();
                    break;
                    }
          return 0;
}

int CVICALLBACK cbline_scan (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];

          switch (event)
                    {
                    case EVENT_COMMIT:
                    GetCtrlVal(ScanGenpanel, PANEL_LINE_SCAN ,&line_scan);
                    if(line_scan==1)
                                        SetCtrlAttribute (ScanGenpanel,PANEL_FRAME_NUM , ATTR_DIMMED,1 );
                               else
                                        SetCtrlAttribute (ScanGenpanel,PANEL_FRAME_NUM , ATTR_DIMMED,0 );


                    if(startScanfg==1){
                    mode=7;                                                     //Start/stop frame scan
                    val1[0]=SCANNERPIC | (address <<1);
                    val1[1]=mode;
                    if(line_scan==1){
                    val1[2]=1;                                                  //1=disable frame scan
                    }
                    else{
                              val1[2]=0;                                        //0=enable frame scan
             }
                              GCI_writeI2C(6, val1, bus);
                    }
                              break;
                    }
          return 0;
}

int CVICALLBACK cbframe_num (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
int val;

          switch (event)
                    {
                    case EVENT_COMMIT:
                    GetCtrlVal(ScanGenpanel, PANEL_FRAME_NUM ,&val);
                    break;
                    }
          return 0;
}

int CVICALLBACK cbscan_enable (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];
int val;

          switch (event)
                    {
                    case EVENT_COMMIT:
                    GetCtrlVal(ScanGenpanel, PANEL_SCAN_ENABLE ,&val);
                    mode=9;                                                     //Enable/disable scanner
                    val1[0]=SCANNERPIC | (address <<1);
                    val1[1]=mode;
                    val1[2]=val;                                                //Enable=0 standby=1
                    GCI_writeI2C(6, val1, bus);
          if(val==1){                                                           //Scanners on standby
                    StartStop(Stop);                                            //Stop the scanning
                    }
                              break;
                    }
          return 0;
}

int CVICALLBACK cbrev_scan (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];
int val;

          switch (event)
                    {
                    case EVENT_COMMIT:
                    RevScan();
                    break;
                    }
          return 0;
}
```

```
int CVICALLBACK cbtimer (int panel, int control, int event,
                    void *callbackData, int eventData1, int eventData2)
{
char val1[20];
int scannerServoError=1;

          switch (event)
                    {
                    case EVENT_TIMER_TICK:
                    mode=0;                                                               //Set mode to read error signal
                    val1[0]=SCANNERPIC | (address <<1);
                    val1[1]=mode;
                    GCI_writeI2C(2, val1, bus);

                    val1[0]=SCANNERPIC | (address <<1) | 0x01;
          if (GCI_readI2C(2, val1, bus)) return -1;                                       //Problem
                    scannerServoError = val1[0] & 0xff;
                    if(scannerServoError==1){                 //Scanner servo error
                               SetCtrlVal(ScanGenpanel, PANEL_SCAN_ERROR_IND ,1);
                                         SetCtrlAttribute (ScanGenpanel,PANEL_START_SCAN , ATTR_DIMMED,1 );
                              if(scanOKfg==1){
                                                  StartStop(Stop);                        //Stop the scanning
                                                  scanOKfg=0;                             //Reset flag
                                                  startScanfg=0;                          //Reset flag
                                           }
                                         }
                    else{
                              SetCtrlVal(ScanGenpanel, PANEL_SCAN_ERROR_IND ,0);
                              scanOKfg=1;                                                 //Set flag
          if(startScanfg==0)
                    SetCtrlAttribute (ScanGenpanel,PANEL_START_SCAN , ATTR_DIMMED,0 );
                    }
                    break;
                    }
          return 0;
}
```

The test user interface panel is shown in Figure 14. Here we use a subset of all the programmable features: the zoom setting is restricted to oscilloscope type settings, and six scanning speeds are catered for. The scan shift controls are just sliders and the scan lag compensation is entered manually.

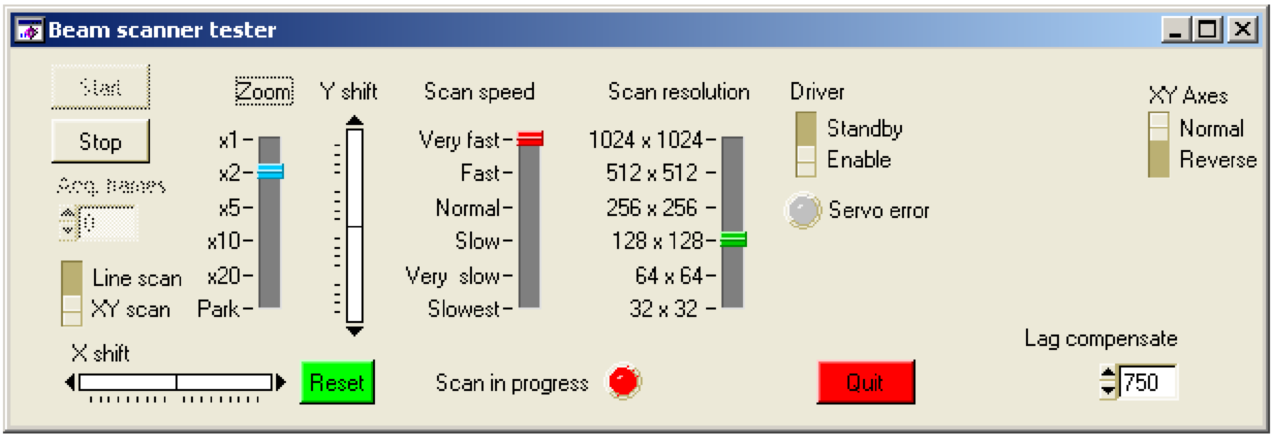
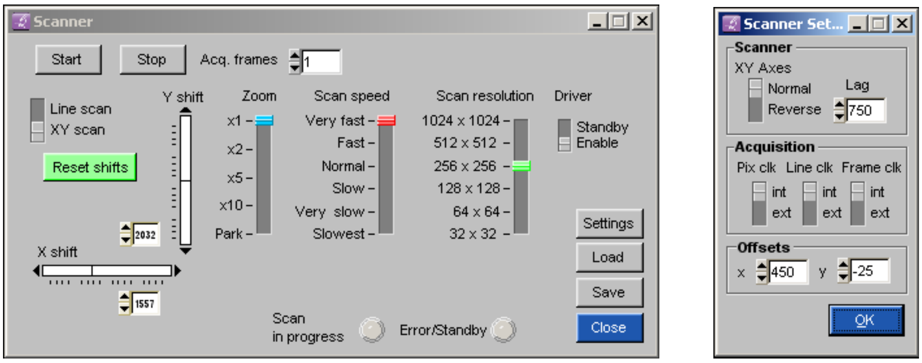


Figure 14. The user interface panel used for instrument testing, developed using LabWindows

The above code allows stand-alone operation of the device and is used during testing and alignment. When this system is used as part of a more complex project (e.g. when incorporated into microscope system, the code and the user interfaces are modified somewhat and are shown in Figure 15. In this instance the code has been modified to provide an application programming interface (API) such that most operations are performed by an even higher level (i.e. laser scanning microscopy). The user interface for direct access to the scanner is provided by a panel much like the stand-alone version but with some settings hidden away from the normal user. The ability to save and load from a settings file is also provided.


Acq. frames 1

Lag 750

Offsets x 450 y -25

2032

1557

Figure 15. The user interface panel used in higher level software is very similar to the stand-alone version but has some settings hidden away in a settings panel which also allows selection of pixel, line and frame clocks from other sources. It also allows for saving and loading of the settings to and from a file.

This note was initially prepared in 2006 and updated during August ad September 2011. B Vojnovic, RG Newman and PR Barber contributed to this note and RG Newman constructed most, but not all the units. Two of the complete systems were put together by Dr D. Matthews of King's College, London, guided by RG Newman.

We acknowledge the financial support of Cancer Research UK, the MRC and EPSRC.