Adwin GUI Design Notes

By: Lindsay LeBlanc, 11 July 2008

CONTENTS

Types of files: 2
Project files (.prj) 2
Workspace files (.cws) 2
Header files (.h) 2
Command files (.c) 2
User Interface files (.uir) 3
Executable files (.exe) 3
Data transfer files (.TA1, .TB1) 3
Important files and what they do 3
vars.h 3
main.c 4
GUIDesign.c 4
- CVICALLBACK CMD_RUN_CALLBACK: 4
- CVICALLBACK TIMER_CALLBACK: 4
- RunOnce 4
- BuildUpdateList 4
- CalcFcnValue 6
- DrawNewTable 6
- SetChannelDisplayed 6
- ReshapeAnalogTable 6
- ReshapeDigitalTable 6
- SaveSettings 6
- SaveArrays 6
- LoadLastSettings 7
- LoadArrays 7
- CVICALLBACK ANALOGTABLE_CALLBACK 7
- UpdateScanValue 7
- CVICALLBACK INSERTCOLUMN_CALLBACK (similar to DELETECOLUMN, PASTECOLUMN, COPYCOLUMN functions of a similar name) 7
- findLastVal 7

Adwin.c 8
AnalogControl.c 8
AnalogSettings.c 8
DACControl.c 8
DACtranslator.c 8
- Reset_timeDAC1220 8
- Reset_bitsDAC1220 8
- SetDACNormal 8

- CalibrateDAC 9
- SetDACVoltage 9

DDSControl.c 9
DDSSettings.c 9
ddstranslator.c 9
DigitalSettings.c 9
LaserControl.c 9
LaserSettings.c 9
LaserTuning.c 9
NumSet.c 10
Processor.c 10
scan.c 10
ScanPanelLoader.c 10
TransferData_TAX_[Date].BAS 10

- AnalogWrite 11
- DDSXWrite 11
- DACWrite_reset 11
- DACWrite_register 11

Updates: 11
Early July 2008: 11

Types of files

Project files (.prj)
This is a file which links all of the files in the project. It's this you should open when starting to modify the code.

Workspace files (.cws)
This is linked somehow to the project file (I don't really understand), but it opens, too, when you open the project file.

Header files (.h)
These files contain all of the function declarations, and some of the variables, especially global variables, used in the project. Certain header files are generated by CVI/LabWindows and should not be modified. There's a message at the top of these files telling you this. These files are created when the User Interface files are generated (see below).

Command files (.c)
These files contain all of the actual programming done by you, the programmer! They also contain the CALLBACK functions generated by the User Interface files, again, see below. The include statements at the tops of these files allow you to link several of them together; this is often done so that the code is somewhat more modular.

User Interface files (.uir)
These files are special to LabWindows, and allow us to create the graphical user interfaces (GUIs) that are so nice to work with. Here, you can lay out a "panel" with input boxes, lists, and/or buttons. Double-clicking on these allows you to change their properties; in addition, you can add or modify the callback functions associated with these, which are functions written in the .c files that allow you execute specific code when an action is taken (things like mouse clicks, data entry, etc.) To generate the callback functions – these are mere skeletons of what you need, go to Code → Generate and choose the appropriate options. The callback functions will appear in which ever .c file was last active unless you specify otherwise. The .h files will be generated with the same base name as the .uir file.

Executable files (.exe)
After compiling the .c files, an executable is created, either in either "debug" or "release" version. This is the file you load when you want to run the experiment. There are certain other files that need to be in the directory for this to work; I'm not sure which ones, and so far have relied on the method of keeping everything around. May try to clean that up sometime soon.

Data transfer files (.TA1, .TB1)
These files are called by the .c files (in particular, by GUIDesign.c) to send commands to the ADwin system. They are written in the special language, ADBASIC, and are linked to the .c code through the Adwin.h and Adwin.c, files which were given to us by the company. The .TA1, .TB1 files are created when you compile and Build → Make bin file.

Important files and what they do

vars.h
This is a header file which contains all of the global variables, constants, and struct definitions used throughout the project. Some (not all) important variables include:

- The "tables" (AnalogTable, DigTableValues, , ddstable, dds2table, dds3table, DACtable, LaserTable). These globally defined arrays, which can be accessed by any function in the project, contain the values of the parameters, and are 2 or 3 dimensional arrays, giving values for each time step, channel, and page.
- Property arrays (AChName, DChName, LaserProperties) for storing settings information for each of the channels
- Structure definitions for each of the above, such that arrays may be defined and many different characteristic may be stored for each position in the array (eg, voltage, ramp style)
- Scanning variables, for storing information that changes with each experimental cycle.
- Global definitions (NUMBERANALOGCHANNELS, NUMBERDIGITALCHANNELS, etc) that define things like the how many times we perform various loops throughout the project.

NOTE: Array definitions differ between C and ADBASIC – C arrays begin at index = 0, which ADBASIC arrays and the user-interface table pointers begin at index = 1. To account for this, all arrays are defined one element larger than they need to be in C, and we consider the index = 0 elements to be useless. In this way, we begin loops at index = 1 to NUMBERANALOGCHANNELS, for instance, and access AnalogTable[index].

main.c

This is the first file in the hierarchy. It contains the commands which are executed when the executable is launched. This includes initialization of all of the arrays and calls for the drawing of the first, blank, panel (though the drawing function is in GUIDesign.c). It then tells the program to begin running through the function *RunUserInterface ( )* and then allows the program to respond to actions by the user through the callback functions.

GUIDesign.c

This file contains the majority of the code in this project. I imagine it was initially created as a file that contained all of the callback functions from the user interface file GUIDesign.uir – it still contains most of these, but other functions have been added to deal with the data. I will list a number of these functions and what they do:

- CVICALLBACK CMD_RUN_CALLBACK:
  This is the function that is called when the start button is pushed. It initiates the experimental cycle by initiating the CVICALLBACK TIMER_CALLBACK function, if the repeat button is pressed, and then calling the RunOnce( ) function.

- CVICALLBACK TIMER_CALLBACK:
  Each time an appropriate amount of time has elapsed, this function calls the RunOnce ( ) function, maybe with new parameters if we've chosen to do a scan.

- RunOnce
  This function reads all of the data out of the tables into which it has been entered on the screen, and turns these three-dimensional arrays into one- or two-dimensional "MetaArrays". These MetaArrays basically just take the page information out of the standard data arrays, giving long one dimensional arrays of number corresponding to each of the time steps in the experiment. At the end of this function, BuildUpdateList is called. This function uses the convention that columns with negative times and those which follow a column with a time = 0 on that page are ignored.

- BuildUpdateList
  Takes the information from the MetaArrays and converts this into a format which is convenient for sending data to the ADWIN itself. In particular, it takes the form of three one-dimensional arrays. With the ADWIN, timing is divided into "Event Periods" – a parameter whose length is selectable from the menu of the main GUI.

We create a system of sending data such that for each event period, we can tell the ADWIN how many and which updates to perform.
These three arrays take the following form:

UpdateNum	**ChNum**	**ChVal**
(number of channels to update during this event)	(channel number that needs updating)	(updated value)
4	2	4.0
	6	1.11
	101	1000 1011 0010 0011
	51	3
1	2	4.1
1	2	4.2

UpdateNum is a list of the number of updates to take place in each event. ChNum tells us which channels should be updated during an event period and ChVal stores the values of those updates. ChNum and ChVal will have the same length, while Update Num will be shorter.

For analog channels (ChNum = 1 to 40), the value of the voltage to be output by the ADwin is ChVal. For digital channels, we write the output in blocks of 16 bits, that is, we update 16 channels at a time, creating this update value by finding the sum of 2^((Digital channel number) mod 16) for all digital channels in the 16 bit block.

"Special" values of ChNum give instructions to deal with non-analog output:

- 101, 102: the lower and upper 32 bit digital output cards (modules 2, 'original' and 3, 'new', respectively)
- 51, 52, 53: write special instructions to DDS1, DDS2, or DDS3, as given by DDStranslator. Information is sent in 2-bit bundles (0, 1, 2, or 3) for serial programming to the DDSs. Values 5 and 6 indicate special instructions for reset.
- 61: reset signal for DAC, information sent in one-bit bundles (ie, send either 0 or 1)
- 62: serial data for DAC, information sent in two-bit bundles (0,1,2,3)

If there is nothing happening for more than one event in a row, a negative number is added to the UpdateNum list. The magnitude of this number is the number of event periods to "skip" – ie, in which to do nothing, while the negative sign acts just as a flag.

The first 999 elements of this list of updates can be output by the GUI to the standard I/O window if, under Preferences, "Print Update Array" is checked.

Finally, the data is sent to the ADwin using the SetPar, SetData_Long, SetData_Float commands which have been provided by ADwin. The timer is started, so that after

the time given by the sum of times in TimeArray has elapsed, the TIMER_CALLBACK will be called again, unless interrupted by a stop.

- CalcFcnValue
  Calculates the ramping features, and determines, given the coarseness of the digital to analog conversion done by the analog cards, if there is indeed a need to update the value of the analog channel in this event period.

- DrawNewTable
  This is the function which is called each time a change is made to the parameters visible on the main panel. It takes as an input either 0 or 1 – with 0, all of the columns stay active, with 1, just those which are non-negative or come prior to a zero column on that page are "activated," ie, are not dimmed. It is here that the colours for the different functions, kinds of cells are specified. This function calls a few other functions: SetChannelDisplayed, ReshapeAnalogTable, ReshapeDigitalTable.

- SetChannelDisplayed
  This function determines whether analog, digital, or both subpanels will be displayed in the main panel, and has some hard-coded numbers that determine their sizes

- ReshapeAnalogTable
  Determines the size and shape of analog table, including the table for the names and units. There are some hard-coded numbers determining the shape in here. Also determines the positions of the DDS offset boxes.

- ReshapeDigitalTable
  Determines the size and shape of digital table, including the table for the names. There are some hard-coded numbers determining the shape in here.

- SaveSettings
  This function is activated when you click on the option in the menu bar, through the callback function CVICALLBACK MENU_CALLBACK, case MENU_FILE_SAVESET. This function saves all relevant data to a .pan file (short, of course, for panel), to be retrieved in the future. It uses the built-in function SavePanelState to save all of the characteristics of the GUI itself, and saves the data separately in a file with the extension .arr (as in, short for 'array') using the functions, SaveArrays, and SaveLaserData in the .laser files.

- SaveArrays
  This function uses the fwrite command to save all of the data arrays (except the laser data) to a file with the same name as the .pan file, saving each variable marked with a buffer name the same as the variable name. In particular, the analog and digital 3-D arrays of values, the analog and digital 2-D arrays of properties, the DDS 2-D arrays of values, and the DAC 2-D array of values are saved. In addition, the update period and DDS settings are saved in this file.

- LoadLastSettings
  This function is activated when you click on the option in the menu bar, through the callback function CVICALLBACK MENU_CALLBACK, case MENU_FILE_LOADSET. This function retrieves all relevant data from the .pan file, using the built-in function RecallPanelState, and user-defined functions LoadArrays, and LoadLaserData in to get back the data saved in SaveSettings / Save Arrays.

- LoadArrays
  Does exactly the opposite of SaveArrays, using the fread function.

- CVICALLBACK ANALOGTABLE_CALLBACK
  To talk about at least one of the callback functions, I take this as an example. This function activates upon a left double-click somewhere in the analog table. The position of the click (defined by a grid quantized by cell number) is read out and depending on where in the table the click happened, a certain secondary panel will be chosen. The secondary panel handles, all of which were identified at the beginning of main.c, are loaded using the built-in DisplayPanel function, which launches the user-interface defined by the handle value assigned to the variable panel_to_display. Once the secondary panel is opened, instructions will be given based on the actions taken by the user upon that panel, in a similar callback sequence.

- UpdateScanValue
  This function is called by CVICALLBACK TIMER_CALLBACK if the Scan button was pushed to start the experimental cycle. There are a few options that could happen within this function. First, if the value passed as an argment is true, it means it's the first cycle and you should use the original variables. If it's false, then the values either come from the list input in the chart at the bottom right of the main panel, or from parameters input in the panel that pops up from the menu. For the list, it uses values that have been saved to the structure ScanVal by the callbacks in ScanTableLoader.c. From the secondary panel, values come from the callbacks in scan.c. The value it determines is then saved to the appropriate globally defined data array, so it can be used upon the next timer event. The simplified data is saved in the variable ScanBuffer, such that it can be later output to a file, using ExportScanBuffer, which comes at the end of the function . When the scan is finished, the values are set back to the original ones.

- CVICALLBACK INSERTCOLUMN_CALLBACK (similar to DELETECOLUMN, PASTECOLUMN, COPYCOLUMN functions of a similar name)
  This function is called when you choose insert column from the menu. It takes the information from your last click in a column and inserts one there, either a copy or blank. This uses the ShiftColumn3 function, which explicitly just copies elements in the arrays from one column to another. If you wanted zeros instead of a copy, it sets all of the elements to zero.

- findLastVal

Finds the previous value of the channel in question, to be displayed in the callback secondary panel boxes, and for comparison to current values when deciding whether it should be changed. Looks explicitly into each kind of data. It makes certain to check for zeros in preceeding columns and will go back a page if it needs to. Also note that it calls itself implicitly sometimes.

Adwin.c
This file contains the functions that communicate with the ADBASIC code the runs the ADwin hardware. This is not something that would be modified.

AnalogControl.c
This file contains all of the callback functions required to deal with actions which arise when the AnalogControl.uir panel is loaded. The main purpose is to write values to the AnalogTable structure, that is, if you double click in a cell, and change a value, these functions update that element of the 3-D array.

AnalogSettings.c
This file contains all of the callback functions required to deal with actions which arise when the AnalogSettings.uir panel is loaded. This happens when you go to Analog Settings in the menu of the main panel. It allows you to change the values of the AChName array, things like the channel name, number, max and min, bias and proportionality values.

DACControl.c
This file contains a single callback function to toggle the state of the global DAC_Status variable to tell other functions whether or not to use send commands on the DAC Channels.

DACtranslator.c
This file contains functions which take information from the DACtable array and turn that into digital data that can be sent to the DAC1220EVM device, through 2 bit commands, which are interpreted by the ADBASIC code.

- Reset_timeDAC1220
  This function determines how much total time the reset sequence will take.

- Reset_bitsDAC1220
  This function determines the entire sequence of 1-bit commands to send to the SCLK to reset the DAC1220

- SetDACNormal
  This function determines the stream of two-bit data-packets (12 of them) to send to the serial input of the DAC1220 to put it in normal mode. It calls other functions defined in this file, NormalModeDAC1220, which hard codes the three bytes to send

and ByteTo2Bits, which turns an integer "byte" value (<= 2^8) into four 2-bit values (<= 2^2).

- CalibrateDAC
  This function determines the stream of two-bit data-packets (12 of them) to send to the serial input of the DAC1220 for self-calibration. It calls other functions defined in this file, CalibrateDAC1220, which hard codes the three bytes to send and ByteTo2Bits, which turns an integer "byte" value (<= 2^8) into four 2-bit values (<= 2^2).

- SetDACVoltage
  This function determines the stream of two-bit data-packets (16 of them) to send to the serial input of the DAC1220 to set the output. It calls other functions defined in this file, SetVoltageDAC1220, which hard codes the four bytes to send and ByteTo2Bits, which turns an integer "byte" value (<= 2^8) into four 2-bit values (<= 2^2).

DDSControl.c
This file contains all of the callback functions required to deal with actions which arise when the DDSControl.uir panel is loaded. This happens when you go to double-click in a DDS row in the main panel. It allows you to change the values of the dds(2,3)table array, including final frequency and amplitude.

DDSSettings.c
This file contains all of the callback functions required to deal with actions which arise when the DDSSettings.uir panel is loaded. . This happens when you go to Analog Settings in the menu of the main panel. It allows you to change the values global variables like the DDSFreq structure and DDSCLOCK variables

ddstranslator.c
This file contains functions necessary for interpreting the data entered into the control panels and outputting two-bit commands for the serial programming of the DDSes. This file remains a mystery to me, but I'm sure whomever is reading this more than has the intellectual capacity to understand it. Go you.

DigitalSettings.c
This file contains all of the callback functions required to deal with actions which arise when the DigitalSettings.uir panel is loaded. This happens when you go to Digtial Settings in the menu of the main panel. It allows you to change the values of the DChName array, things like the channel name, number, and reset properties.

LaserControl.c
LaserSettings.c
LaserTuning.c
These files contain all of the callback functions required to deal with actions which arise when the LaserControl.uir, LaserSettings.uir, and LaserTuning.uir panels are loaded. I

haven't spent any time working with these and don't know too much about what goes on in here.

.

NumSet.c

This file is associated with the ScanTableLoader.c file, and contributes to loading values to the scan list. Don't know much more.

Processor.c

This file contains callback functions to toggle between processors, so that the code will work with either ADwin we own. Sets the global variable processorT1x.

scan.c

This file contains all of the callback functions required to deal with actions which arise when the scan.uir panel is loaded. This panel is necessary for setting up a scan, and depending on which button is pressed, it will perform that kind of scan. Numerical values given to each kind of scan are: 0: analog magnitude, 1: time scan, 2: DDS end-frequency scan, 3: DDS floor scan, 4: Laser scan, 5: DAC setvoltage scan. In general, this function loads values into the PScan global struct variable by reading values in from those entered into the panel.

ScanPanelLoader.c

This file contains all of the callback functions required to load numbers into the Scan Values table, to be read in GUIDesign.c, and used from there.

TransferData_TAX_[Date].BAS

eg.) TransferData_TA1_09July2008.BAS

This file is the code which talks to the ADwin hardware. It gets as inputs the 3-columns produced by the BuildUpdateList function, and uses this information to send data to the ADwin.

I don't know too much about the LED setting functions, but they're more important for debugging purposes.

This file receives data from the BuildUpdateList function, as described above, and saves it in three arrays, Data_1 (UpdateNum), Data_2 (ChName), and Data_3 (ChVal) (not case-sensitive). It also gets other parameters, like the update period. To begin, variables are initialized and the hardware is configured in the INIT: section of the code.

The EVENT: section of the code is the most important; it is the main function. This section will repeat regularly with a period given by GlobalDelay. It is able to deal with the negative numbers in Data_1, and changes the delay that period to wait the extra time when nothing happens. If there is a positive number in Data_1, it sets up a loop that runs that number of times, during which each value in Data_2 and Data_3 is checked and the appropriate action is taken. If ChName refers to an analog channel (1 to 40), the AnalogWrite function is called, if DDS (5X), then DDSXwrite is called, if digital (10X),

then the values are written to the digital cards using DIG_WRITELATCH32, and if DAC (61, 62), either DACWrite_reset or DACWrite_register are called.

When EVENT is finished, check for final zeros and reset everyone.

- AnalogWrite

  Determines which analog card to write to, then uses writedac to write the appropriate value to the appropriate channel on the appropriate card.

- DDSXWrite

  Takes the 2 bit code, and writes one bit at a time to the appropriate digital lines, which have been reserved for this purpose. Some of the commands are repeated, for timing reasons, and this function has been optimized to work with 10\mus update periods, though the DDS should work for longer update periods as well. Data is written to the dataline and must be accompanied by signals to the clock line. This function also checks for the ioupdate and master reset possibilities, which come in the form of codes 5 and 6.

- DACWrite_reset

  This function will tell the SCLK line to be either high or low for the duration of the EVENT period, depending on the parameter sent to this line. Because DIG_WRITELATCH32 must be used to maintain the value at the end of the EVENT cycle, we must also consider the last value written to that bank of digital channels and make sure the same thing is written to all of those channels as we update SCLK.

- DACWrite_register

  Similar to the DDSXWrite functions, takes a 2 bit code and writes each bit in serial form to the SDIO line, while clocking the SCLK line. DIGOUT_F functions are repeated to get the pulses to be sufficiently long to properly trigger the DAC1220. (This was done rather by trial and error).

Updates:

Early July 2008:
Added the capability to output a digital stream that is able to control the DAC1220 I bought, a 20-bit digital to analog converter that will allow for a precise signal to be sent to a feedback loop, in particular, for controlling the Feshbach current.

In general, the program is a mess. I want to write down some of what I've learned from it, in case others (or I) ever again need to go through this ordeal.