СОЗДАНИЕ СОБСТВЕННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ ЦИФРОВОГО ОСЦИЛЛОГРАФА

Суханов Е.В.

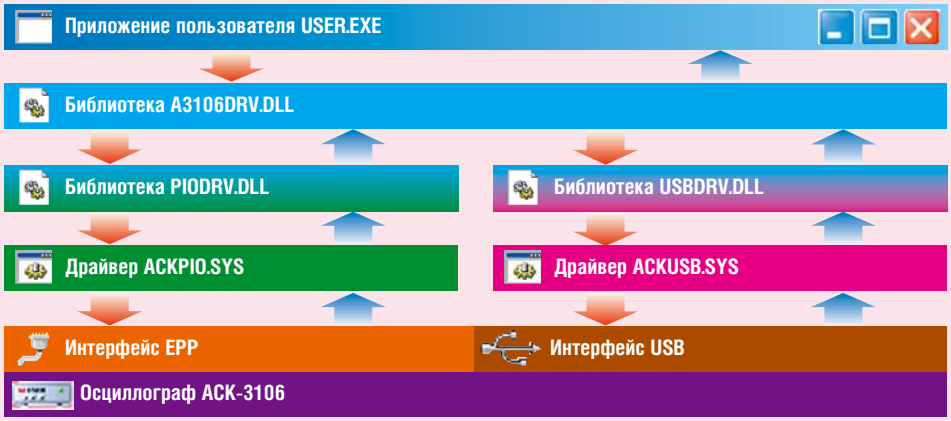
Читатели КИПиС уже знают (см. КИПиС, № 4, 2003 г.) о цифровом запоминающем осциллографе АКТАКОМ АСК-3106 и о прилагаемом к нему фирменном программном обеспечении (ПО). Но при всех достоинствах фирменного ПО у него есть один неустранимый изъян: оно обязано подходить большинству пользователей. А универсальность — всегда враг эффективности. Поэтому, если Вы планируете использовать прибор для решения какой-либо единственной специальной задачи, фирменная программа может оказаться либо слишком громоздкой и медлительной, либо вовсе неприменимой. В этом случае Вам может помочь предлагаемый производителем комплект разработчика ПО для осциллографов серии АСК-310х.

Этот комплект разработчика (по традиционной международной терминологии — DK, Developer Kit) содержит полный набор драйверов, библиотек функций, документации и примеров программирования, требующихся для «приручения» прибора. Создайте с помощью DK собственную программу — и осциллограф будет делать то, и только то, что Вам необходимо (конечно, если поставленная задача вообще ему по силам). Но отложим пока решение конкретных задач, и рассмотрим для начала, какие средства для их решения входят в состав комплекта.

Первое, что должен обеспечить комплект разработчика — это возможность передачи команд управления из программы пользователя в прибор, и обратный поток данных. В данном DK передача информации между программой и прибором осуществляется с помощью следующих системных драйверов и динамических библиотек:

- драйвер портового ввода-вывода AckPIO.sys, который обеспечивает связь с прибором через интерфейс EPP;
- драйвер контроллера USB AckUSB.sys, который обеспечивает связь с прибором через интерфейс USB;
- библиотеки портового ввода-вывода PIODrv.dll и контроллера USB USBDrv.dll, которые содержат функции для работы с драйверами AckPIO.sys и AckUSB.sys соответственно; необходимы для работы библиотеки a3106drv.dll;
- библиотека функций управления прибором a3106drv.dll, которая содержит все необходимые функции для работы с прибором, включая управление драйверами и функции низкого уровня и сопровождается заголовочным файлом для C/C++.

Иерархия взаимодействия этих драйверов и библиотек представлена на схеме (см. рис.).



Как видно из схемы, Ваше приложение непосредственно будет взаимодействовать только с библиотекой a3106drv.dll, которая избавляет программиста от необходимости работы на нижних уровнях. И в документации, прилагаемой к DK, подробно описывается только использование этой библиотеки. Рассматриваются как все глобальные определения, структуры и переменные библиотеки, отдельные функции, так и последовательности вызовов функций библиотеки для решения типичных задач.

Кроме библиотек, драйверов и документации в состав DK входит также пример полностью функционального приложения, иллюстрирующего способы работы с осциллографом при помощи средств DK. Пример написан в среде Borland C++ Builder 6 и предлагается как проект именно для этой среды. Однако те участки исходного текста программы, которые отвечают непосредственно за управление прибором и чтение данных, написаны без использования особенностей Borland C++ и даже ANSI C++, так что Вы можете использовать их без изменений в любом компиляторе C/C++.

В примере имеются элементы, позволяющие полностью управлять прибором в осциллографическом режиме и наблюдать полученные осциллограммы. Изучив предлагаемый пример, Вы легко сможете воспроизводить аналогичные действия с прибором в своей собственной программе.

Теперь попробуем решить с помощью DK такую достаточно типичную производственную задачу.

Пусть для контроля некоторого процесса необходимо в определенные моменты времени снимать осциллограмму, сравнивать ее с эталонным образцом, результаты измерений записывать в архив, а в случае критического отклонения контролируемых параметров дополнительно подавать аварийный сигнал.

При всей простоте задачи, ее крайне сложно решить с помощью стандартного программного обеспечения АСК-3106. Зато с помощью DK решение быстро оформляется в виде короткой и эффективной программы. Действительно, все измерения проводятся при заранее известных настройках прибора, которые во время работы нет необходимости изменять, и, следовательно, нет необходимости создавать средства пользовательского интерфейса для управления прибором. Нет необходимости также в ручных курсорных измерениях или в автоматических измерениях лишних параметров, не имеющих значения для данного процесса. По сути, нет необходимости и в изображении измеряемых осциллограмм — весь анализ программа может провести в математической форме, не отвлекаясь на визуализацию. Весь интерфейс пользователя может состоять из двух кнопок («Старт» и «Стоп») и двух индикаторов («Годен» и «Не годен»). Не намного сложнее оказывается и написание внутренних алгоритмов программы. Все блоки, отвечающие за работу осциллографа, можно взять из готовых примеров DK, причем и их можно упростить, исключив изменение фиксированных настроек прибора. Таким образом, программист может сосредоточиться только на модуле анализа измерений, который и будет представлять специфику решаемой задачи. Текст итоговой программы в общем виде показан во врезке.

Здесь предполагается, что модуль анализа измерений будет содержать четыре функции:

ProcessMessages — функция-обработчик системных сообщений, которая

```
#include <windows.h>
#include <math.h>
#include <stdio.h>
#include "a3106drv.h"
#include "analise.h"
OscilloscopeData * scopeData_ptr; //указатель на параметры осциллографа в драйвере
char key[20];
int
  start_register=0, //флаг запуска измерений
  exit_flag=0;      //флаг выхода из программы
double
  nonius,           //текущее значение нониуса (в секундах)
  min_nonius,       //минимальное значение нониуса (в кодах)
  max_nonius;       //максимальное значение нониуса (в кодах)
BYTE                //буфера данных
  BufferA[DATA_DIM], BufferB[DATA_DIM];
/////////////// Прототипы функций ////////////////////////
void ResetScope (int force); //переустановить параметры осциллографа
// Прочитать ключ доступа из файла
int ReadKeyFile (char *key, char *fname);
void InitPrg (void); //пользовательская инициализация программы
int ReadSignal (void); //снятие измерений
int DataRegister (void); //цикл измерений и отображения сигнала
int SetDefaultCfg (void); //установить фиксированные настройки прибора
//-------- из пользовательского модуля analise ----------//
int ProcessMessages (void); //обработать очередные системные сообщения
int AnalyseFunc (void); //проанализировать осциллограммы
int WriteFunc (void); //записать на диск результаты
int IndicateFunc (void); //индикация результатов и аварийная реакция
//переустановить параметры осциллографа
void ResetScope (int force)
{
  SetTimeBase (scopeData_ptr->timeBase );
  SetTriggerDelay (scopeData_ptr->trgDelay);
  SetPosttriggerLength (scopeData_ptr->postTrgLength);
  if (force) {
    SetRange (scopeData_ptr->rangeA, IDC_A );
    SetRange (scopeData_ptr->rangeB, IDC_B );
    SetOffset (scopeData_ptr->offsetA, IDC_A);
    SetOffset (scopeData_ptr->offsetB, IDC_B);
    SetSubOffset (scopeData_ptr->suboffsetA, IDC_A);
    SetSubOffset (scopeData_ptr->suboffsetB, IDC_B);
    SetTriggerLevel (scopeData_ptr->triggerLevel);
    SetTriggerSource (scopeData_ptr->triggerSource);
    SetTriggerLogic (scopeData_ptr->triggerLogic );
    SetScrollMode ( 0 );
  }
}
//установить фиксированные настройки прибора
int SetDefaultCfg (void)
{
  scopeData_ptr->timeBase = 4;
  scopeData_ptr->trgDelay = 0;
  scopeData_ptr->postTrgLength = 10000;
  scopeData_ptr->rangeA = 5;
  scopeData_ptr->rangeB = 5;
  scopeData_ptr->ACDC[IDC_A] = CPL_AC;
  scopeData_ptr->ACDC[IDC_B] = CPL_AC;
  scopeData_ptr->triggerLevel = 127;
  scopeData_ptr->triggerSource = IDC_E;
  scopeData_ptr->triggerLogic = LEADING_EDGE;
}
// Прочитать ключ доступа из файла
int ReadKeyFile (char *key, char *fname)
{
  FILE *stream;
  stream = fopen (fname, "r");
  if (stream == NULL) return -1;
  fscanf (stream, "%s", key);
  fclose (stream);
  return 0;
}
//пользовательская инициализация программы
void InitPrg (void)
{
  int err;
  InitOscilloscope (); //инициализировать параметры осциллографа
  //получить указатель на структуру данных прибора
  scopeData_ptr = GetOscilloscopeDataPtr ();
  ReadKeyFile (key, "key.txt"); // Прочитать ключ доступа из файла
  AccessKey (key); // Передать в драйвер ключ доступа
  //проверить связь
  if ((err = TestCommunication ()) != E_NO_ERROR)
    MessageBox(NULL, GetDRVErrorMsg(err), "Ошибка", MB_OK|MB_ICONWARNING);
  else {
    MessageBox(NULL, "Связь с прибором успешно установлена", "OK", MB_OK|MB_ICONINFORMATION);
    //переслать настройки в прибор
    ResetScope (1);
    //попытаться откалибровать нониусы
    if (CalibrateScope (&min_nonius, &max_nonius, 100) == E_NO_ERROR)
      good_nonius = ((max_nonius-min_nonius) > 40);
  }
  //установить фиксированные настройки прибора
  SetDefaultCfg ();
  exit_flag = 0;
}
//снятие измерений
int ReadSignal (void)
{
  BYTE status;
  int signal_size, res;
  res = -1;
  scopeData_ptr->start = 0;
  ResetScope (0); //"мягкая" переустановка параметров осциллографа
  StartRegistration (1); //команда запуска регистрации
  //цикл ожидания начала регистрации
  do {
    ProcessMessages(); //обработать очередные сообщения
    if ((!start_register) || exit_flag) goto ERR; //пользователь прервал процесс?
    ReadStatus (&status); //чтение статуса прибора
  } while ((!(status & STATUS_START_READ_DATA)) && status); //пока не начнется регистрация
  do {
    ReadStatus (&status);
    ProcessMessages();
    if ((!start_register) || exit_flag) goto ERR;
  } while (status & (STATUS_START_READ_DATA|STATUS_DELAY_END)); //пока не закончится регистрация
  StartRegistration (0); //команда остановки регистрации
  //размер сигнала = задержка запуска + послезапись
  signal_size = scopeData_ptr->trgDelay + scopeData_ptr->postTrgLength;
  //но не больше размера буфера
  if (signal_size > DATA_DIM) signal_size = DATA_DIM;
  //пользователь еще не выключил измерения?
  if (start_register)
    ReadData (BufferA, BufferB, signal_size); //прочитать данные из буфера прибора
  res = 0;
ERR:
  return res;
}
//цикл измерений и отображения сигнала
int DataRegister (void)
{
  while (start_register && (!exit_flag)) {
    ReadSignal (); //снять измерения
    AnalyseFunc (); //проанализировать осциллограммы
    WriteFunc (); //записать на диск результаты
    IndicateFunc (); //индикация результатов и аварийная реакция
    ProcessMessages(); //обработать очередные сообщения
  }
  return 0;
}
```

должна обеспечивать передачу в программу команд пользователя, устанавливая глобальные флаги start_register и exit_flag;

AnalyseFunc — функция-анализатор осциллограмм сравнивающая полученные данные с эталоном;

WriteFunc — функция для записи архивов измерений на диск;

IndicateFunc — функция для индикации результатов анализа и, при необходимости, для реакции программы на ситуации критического отклонения измеряемых параметров.

Приведенный пример программы с использованием DK достаточно наглядно демонстрирует эффективность использования тандема «прибор + DK» практически для всех случаев нестандартного использования этого прибора в качестве устройства сбора аналоговых данных. Использование этого тандема позволяет значительно расширить область применения прибора, т. к. любой пользователь, имеющий достаточные навыки программирования с помощью DK легко и оперативно напишет именно ту программу, которая необходима в данный момент под конкретную задачу с необходимым ему интерфейсом. Кроме того, при изменении задачи можно достаточно просто адаптировать под нее тот же самый прибор, изменив только интерфейс пользователя и процедуры обработки собранных данных, не привлекая к этому поставщика оборудования. Это позволяет значительно сократить время (и затраты) на разработку ПО, да и всего измерительного комплекса в целом.

The Software Development Kit for oscilloscopes AKTAKOM ACK-310x is described in this article. The example of use of this kit is considered.