International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Essentials of Building Virtual Instruments with LabVIEW and Arduino for Lab Automation Applications

Jianghua Bai, Andres La Rosa

Portland State University, the Physics Department, USA

Abstract: ***Four ways to improve the capabilities of a virtual instrument involving a microcontroller are covered in this paper. They are structural modeling and programming, real-time control, asynchronous communication between the microcontroller and the host PC, and system integration. This paper covers 4 common problems encountered by embedded developers and 5 solutions to the 4 problems. The solutions and examples demonstrated in this article will help readers build robust and reliable virtual instruments for crucial applications.***

Keywords: Virtual instrumentation, LabVIEW, Arduino, lab automation, layered models

1. Introduction

Different from amateur electronics for hobbyists, lab automation applications want the system to be stable, reliable, and accurate enough and fast enough. Arduino is an open source system with hundreds of libraries and compatible hardware sets. Arduino makes the development of microcontroller based systems much easier than with traditional register programming methods. Since Arduino lowers the barrier of embedded programming requirement, more and more students, engineers and scientists are employing Arduino in their applications.

By downloading online resources, a lot of simple projects can be accomplished through the Arduino platform. There are also tools, like LINX, MATLAB, or LabVIEW, helping intermediate users develop complex applications [1-2]. But beside of tools, developing complex and critical automated systems with embedded processors involves some other advanced issues and solutions. This paper will go through these issues and solutions to help the readers developing advanced control systems and automated systems.

By evaluating and developing hundreds of embedded and virtual instrumentation projects, we saw many flaws and failures. Compiling the discrepancies, we finished this paper. The purpose of this article is to guide intermediate embedded developers to become advanced embedded and virtual instrument developers.

2. Structural modeling and developing

The first issue is that we have software, hardware, and tasks needed to be done. How to organize the software and hardware resources and fulfill the tasks effectively and efficiently? For a simple project, one just downloads codes from the web, hooks up hardware and then tweak and tune the system. But for a complex project, there may be not available onlinesolutions. This means that you need to develop it from scratch! How to finish the project easily and successfully?

The solution is structural modeling!

Other than, jumping to the programming and connecting your hardware, soon after you get the project, you need to build a model of your system. For detailed modeling and reasoning, one can check Ref [3]. Here, we will cover a 4 layer model.

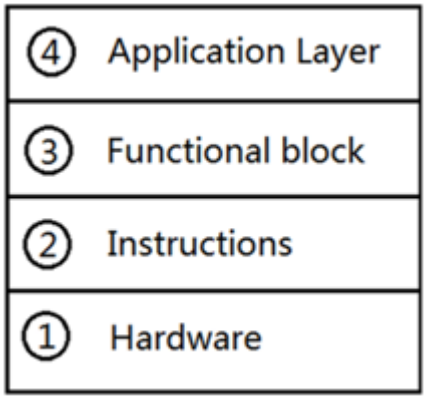

Figure 1: 4-layer model of a complex embedded system or virtual instrument.

Why 4 layers, not 5 layers or a single layer?
A single layer model is usually bad for complex systems. If all software, hardware resources and algorism are organized in a flat model, it is hard to develop and maintain the system. A 5 layer model is usually more than enough for an embedded project. So usually a 3 or 4 layer model is better! For a stand-alone system, a 3 layer model may be enough. But in order to future expansion and the system integration, a 4 layer model is the best. [4]

In Layer 1, one organizeshis hardware carefully, according to hardware mechanical property, the electrical connectors, voltage levels, etc.

In Layer 2, one programs the hardware according to their capabilities and builds instruction set for each job your hardware is capable of doing.

In Layer 3, according to some algorism, timing or procedure, one builds the basic functional blocks with the Layer 2 instructions. In Layer 4, one builds the graphical user interface, the signal processing and data visualization part of the system. By programming the system with a structural model, one can transfer and reuse many of the blocks to a new project. This not only simplifies the developing process but makes it easier to identify any design error. For a complex project, designing a correct model is the first and the most fundamental step. If there is an error in your model, ones' beautiful coding and hardware become useless.

3. LabVIEW + Arduino

The 2nd issue is that there are hundreds of software and hardware available, which combination should beused? One of the solutions is combining LabVIEW with Arduino.

LabVIEW is the de faro standard software for lab measurements and controls. It is easier to integrate or expand the embedded system with some other LabVIEW controlled system to build an even more complex system.

There is an entry-level Arduino and LabVIEW toolkit called LIFA, for free download [2]. But, with this toolkit, one cannot program the Arduino board, instead, one could only program in LabVIEW. For a complex task, one definitely should code his own Arduino applications. Combining the solution 1 and 2 together, one has the system structure shown below.

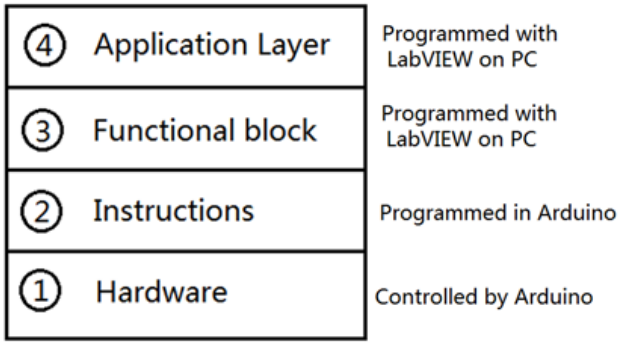

Figure 2: 4-layer model of an Arduino and LabVIEW combined system.

4. Identify the real-time signals

The second solution brings another issue. How to plan the whole project? How to split the functionality between Arduino and LabVIEW. The solution is to plan the jobs by timing! There are real time events and non-real time events. There are real time controls and ordinary on-line controls.

The first step is to identify the real-time events and controls. LabVIEW programs run on PCs are not capable of real-time controls or reactions because the PC usually has a long time delay and bus latency, which cannot meet the critical timing requirements unless the controlled system is really really slow! Some real time events need to be acknowledged immediately when it triggers. The real time events driven jobs should be coded in Arduino at Layer 2. See fig 2. The non-real time events and jobs can be coded in LabVIEW in Layer 3. For non-real time events, the Arduino side only works as I/O ports for the LabVIEW programs.

The next step is to finish the real-time controls in Arduino. A simple block diagram of a closed-loop control system is shown in fig 3. We input r(t) to the Plant and want the Plant to work as r(t). But the Plant has its' own characteristics, in most of thecases, the plant cannot follow the reference signal r(t) directly. One way to make the Plant to follow the requirement of r(t), is to control the Plant through a controller and build the whole system into a feedback system. See fig 3. [5] [6]

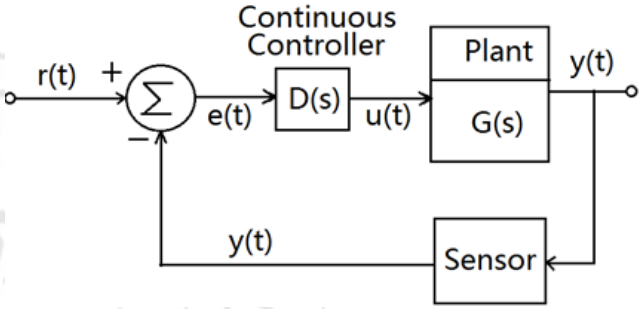

Figure 3: A simple block diagram of a closed-loop control system [5]

Although the world is analog, the Arduino processors are digital. In order to control an analog plant with a digital processor, one builds the control system as the one shown in fig 4. The digital processor samples the input r(t) and the sensor output y(t) and finishes the control schemes with difference equations in the digital domain, then the processor outputs its control commands to the plant through a DAC and actuator. The details about digital controls can be found in Ref [7].

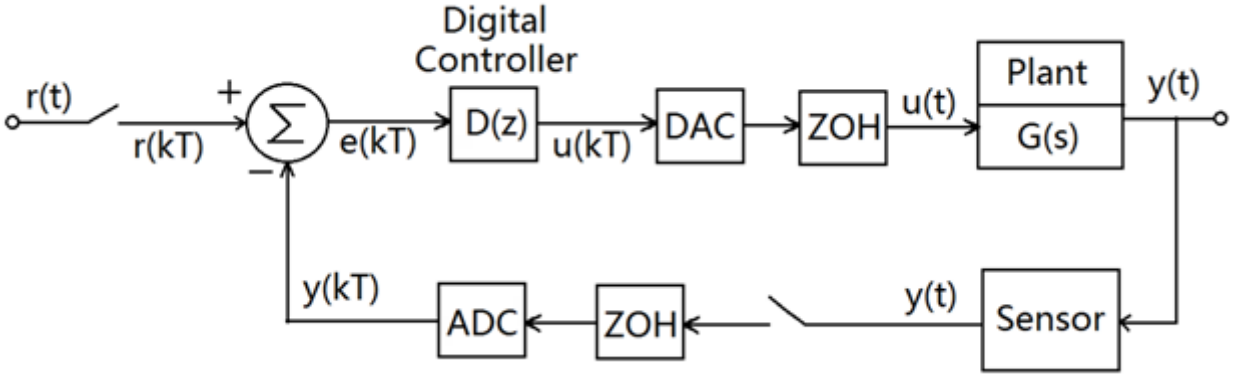

Figure 4: A simple block diagram of a numerical closed-loop control system [5]

Remind, the real-time controls and real-time events must be processed deterministically according to their timing. Otherwise, the whole system just does not work.

5. Asynchronous communication between Layer2 and Layer3

According to the model shown in fig 2, Layer 1 and Layer2 are implemented by Arduino, and Layer 3 and Layer 4 are implemented by LabVIEW! The other issue is how to communicate between Layer 3 and Layer 2! When programming Layer 2 in Arduino, one wants to forget what happens in Layer 3. Similarly, when programming Layer 3 in LabVIEW, one wants to forget what happens in Layer 2! A layered model, allows the programmer to code each layer independently and let adjacentlayers to communicate through interfaces. A layered model greatly simplifies the programming and maintenance of the codes. But, one also wants different layers to work together seamlessly.

The solution is asynchronous communication between Arduino and LabVIEW! In Arduino side, a loop is running to meet the real-time controls. By the same time, Arduino produces the data and save the data in a buffer for Layer 3 procedures to read. A simple illustration is shown in fig 5. In this example, line 20 of the code does the real time data logging and buffer writing. The whole timing is controlled by the while loop. In some critical applications, one may use interrupt to keep the timing and save the data inside an array[8]. When the system is free, the chunk of data in the array can be dumped to the serial buffer at once. This may increase the speed of the control loop.

```
11 void loop() {
12
13   while(1){
14     for(char i=2;i<=13;i++){
15       // in the form CXXD,
16       delay(2),Serial.print('C');
17       if(i<10) Serial.print(0,DEC);
18       delay(2),Serial.print(i,DEC);
19       // next line mimics the real time control and buffer writing
20       delay(2),digitalRead(i)? Serial.print(1):Serial.print(0);
21       //
22       delay(2),Serial.println(',');
23     }
24   }
25 }
```

Figure 5: An example of how to build a data package in Arduino

In the LabVIEW side, a simple finite state machine can be used to implement the communication between Layer 3 and Arduino. Fig 6 shows a simple LabVIEW program used to communicate with Arduino through COM port. Fig 5 and fig 6 are used for illustration purpose. They are not working in apair here.

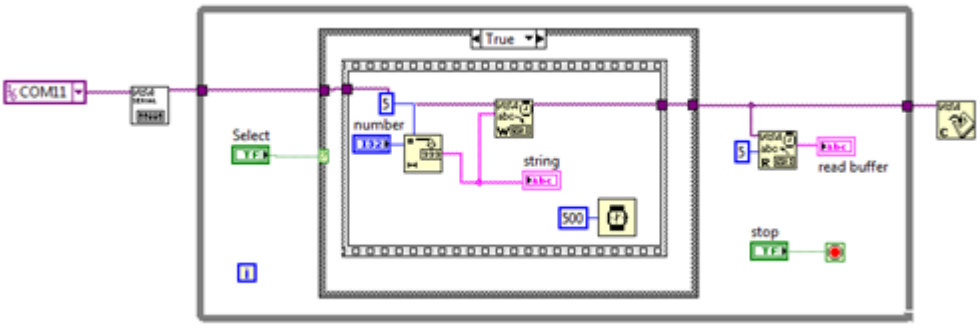

Figure 6: An example of how to talk with Arduino with a LabVIEW coded FSM

Remind, one wants to simplify the math before programming. One can have a complex programming of a complex task, a simple programming of a complex task, or a complex programming of a simple task! The math model controls with which one will end up with. Simplifying the math before programming is the only way leading to a simple programming of a complex task!

6. Build data into a package

When the Layer 3 and Layer 2 programming are done. The system may not work the designer planned. This issue is due to the asynchronous communication between Layer 2 and Layer 3! Asynchronous communication brings the freedom to the programming. But how to guarantee the Layer 2

subroutines in Arduino get the correct commands from LabVIEW and the Layer 3 LabVIEW routines get the correct data from Arduino? The solution is to build the data into a package!

The package always have the same length and the same format! The data package in fig 5 is 5 bytes long. The first byte is the header, 'C'. The second and third bytes are channel numbers. The fourth byte is the data from the relevant channel. The fifth byte is the ending, ','.

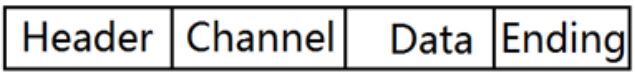

Figure 7: A data package example for serial communication between Layer 2 and Layer 3.

When LabVIEW code reads this data package, it verifies the header and checks the channel and reads the data! The Arduino program in Layer 2 can dump the data at the serial buffer with its own rate. While the LabVIEW program in Layer 3 will read the data from the serial buffer at another rate. Because the data are carefully formatted, LabVIEW codes cannot mess up with the data. Suppose, there were no header, no ending, Arduino would write the channel number and data directly to the serial buffer. LabVIEW programs at layer 3 may mess up with the data, due to asynchronous communication.

Fig 8 and fig 9 show an interface between Layer 3 and Layer 2. In this example, thedata package is 3 bytes long. Each command package starts with 'L' and 'C' and followed with the command, which LabVIEW program wants the Arduino to execute. Both the LabVIEW program and the Arduino program are running at their own speed. When there is a need to communicate, Layer 2 and Layer 3 routines will read or write the serial buffer. Program in Layer 3 does not need to wait for Arduino timing, while the Arduino program in Layer 2 does not need to wait for the LabVIEW timing. Through this example, readers can really enjoy the freedom of asynchronous communications.

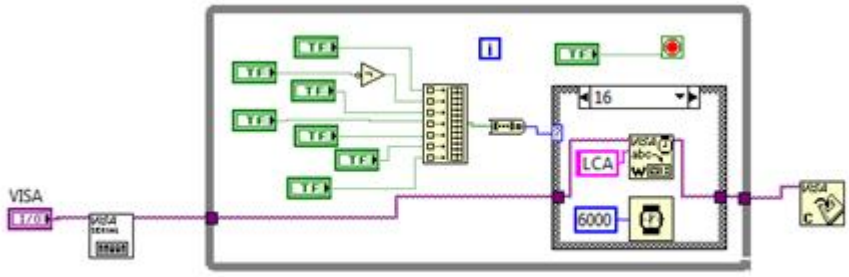

Figure 8: A LabVIEW coded FSM talking to Arduino through COM port

```
167 void loop() {
168 char cmd[3];
169  if(Serial.available()) {
170      for (char i=0;i<3;i++){
171       cmd[i]=Serial.read(),delay(3);
172      }
173       if(cmd[0]=='L'&& cmd[1]=='C'){
174        switch(cmd[2]) {
175      case 0x41:  Low200St3(),digitalWrite(12,HIGH);
176                  break;
177      case 0x42:  Low50St3();
178                  break;
179      case 0x43:  Low10St(123);
180                  break;
181      default:    break; } } } }
```

Figure 9: The paired Arduino code to talk with LabVIEW at Layer 3.

Since the LabVIEW VISA only reads and writes ASCII strings[9]. The programming methods shown in fig 8 and fig 9 is straightforward. If one wants to send and read binary data between Arduino and LabVIEW, please read Ref [10]. Another issue is that the Arduino resource is quite limited. If everything is stored in byte ASCII, it is a waste of Arduino resources. An improvement can be made by storing Arduino states with a single bit other than one byte. This way, each byte can store 8 state information. That makes the storage and data transferring more efficient and faster. Again the Arduino will write the data as ASCII string to the serial buffer. At Layer 3, the LabVIEW routines need to read the string and decode the string properly to get the correct information. Fig 10 and fig 11 show an integer is used to transport 12 channel information at once. Again, Arduino will write the integer as strings in the serial buffer [11]. A 16 bit unsigned integer can represent a number from 0 to 65535, zeros may be inserted when the number is small to keep the data have the same format after they are converted into strings. See fig 10.

```
//==================loop============
void loop() {
  // output format < "4096,"
  while(1){
    for(char i=2;i<=13;i++)
    // save HIGH and LOW state on one bit
    digitalRead(i)? bitSet(Vector,i-2):bitClear(Vector,i-2);
    //
    if   (Vector<10)   Serial.print('0');
    if   (Vector<100)  Serial.print('0');
    if   (Vector<1000) Serial.print('0');
    delay(2),Serial.print(Vector,DEC);
    delay(2),Serial.println(',');
   }
}
```

Figure 10: An example of building binary data packages in Arduino

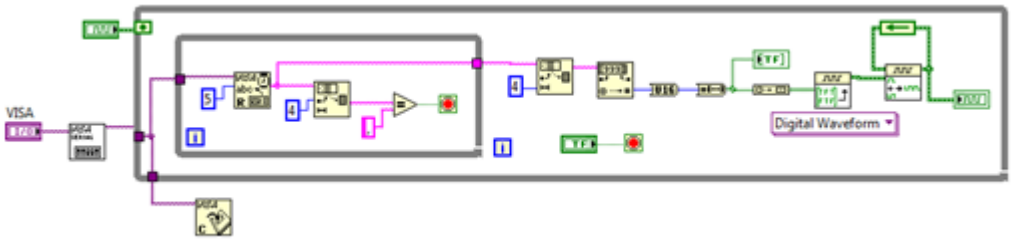

Figure 11: The paired LabVIEW code to receive Arduino binary information

There is no standard to format the data package. In order to make the whole system work, engineers just need to keep the data format ideas in mind and match the interface between Arduino and LabVIEW.

7. Conclusion

Achieving safe, stable, simple coding of complex tasks is the ultimate dream of embedded developers. The 5 issues mentioned in this paper are the most common points leading to a failed or bad embedded system. After solving these 5 major issues with the solutions provided in this article, one will be able to develop a sophisticated virtual instrument system with Arduino boards and LabVIEW.

There are basic rules to building a professional embedded virtual instrument with LabVIEW. In essence, try to simplify your model, before you jump to design and implement your systems. Build a clear structure of your system, is always helpful. Build your system with a 4 layer model. Layer 4 only has necessary human inputs and monitors. In the LabVIEW side, keep the routines in Layer 3. Build a human interrupt and anemergency stop inside of the system, throughout Layer2, Layer 3 and Layer 4.Try to keep each LabVIEW program on one screen!

References

[1] Arduino, Linx, http://playground.arduino.cc/Learning/Linux
[2] National instruments, http://forums.ni.com/t5/LabVIEW-Interface-for-Arduino/ct-p/7008
[3] Kenneth H. Rosen, Discrete Mathematicsand Its Applications, McGraw-Hill companies 2012
[4] Jianghua Bai, Jingwei Chen, John Freeouf, Andres La Rosa, A 4-layer method of developing integrated sensor systems with LabVIEW, Journal of Measurement Science & Instrumentation,2013
[5] Gene Franklin, Feedback Control of Dynamic systems, Pearson Higher Education, Inc. 2010,
[6] Richard Dorf, Modern Control Systems, Pearson Higher Education, Inc. 2015
[7] Katsuhiko Ogata, Discrete-Time Control systems, Pearson Higher Education, Inc. 2015
[8] amandaghassaei,Arduino Timer Interrupts,http://www.instructables.com/id/Arduino-Timer-Interrupts/
[9] NI, Send or Receive Binary or Hexadecimal Data Using NI-VISA in LabVIEW , http://digital.ni.com/public.nsf/allkb/6C24F2F07BC23BB87862572280071086 5
[10] NI, Writing Bits to the Serial Port Instead of Writing ASCII Strings, http://digital.ni.com/public.nsf/websearch/575CDC2EEA251F3086257062007645CA?OpenDocument
[11] Arduino programming references, https://www.arduino.cc/en/Reference/HomePage