Programmable logic devices in experimental quantum optics

John Stockton, Michael Armen, and Hideo Mabuchi

Norman Bridge Laboratory of Physics 12-33, California Institute of Technology, Pasadena, California 91125

Received April 3, 2002; revised manuscript received June 24, 2002

We discuss the unique capabilities of programmable logic devices (PLDs) for experimental quantum optics and describe basic procedures of design and implementation. Examples of advanced applications include optical metrology and feedback control of quantum dynamical systems. As a tutorial illustration of the PLD implementation process, a field-programmable gate array controller is used to stabilize the output of a Fabry-Perot cavity. © 2002 Optical Society of America

OCIS codes: 120.0120, 120.2230, 120.5050, 270.0270.

1. INTRODUCTION

Controllers are pervasive in experimental physics. Servos typically play a role behind the scenes, stabilizing environmental conditions (e.g., temperature, frequency and amplitude of driving lasers) for the physical system of primary interest (e.g., quantum dots and trapped atoms or molecules). But the system of interest can itself be the explicit object of sophisticated control strategies. An increasing number of experimental quantum systems are being developed to the point at which coherent dynamics occur at time scales longer than the response time of available detectors and actuators.[1–3] This separation of time scales opens the door for real time feedback control to be applied in quantum mechanical scenarios.

New theoretical and experimental tools will be required to achieve quantum control objectives. Concerted efforts are currently being made to extend classical control theory to quantum problems in which measurement backaction cannot be ignored.[4,5] Given the inherent nonlinearity of conditional quantum dynamics, optimal control laws cannot be practically implemented with analog circuits, necessitating fast digital control. Even for linear systems, programmable logic may be superior to analog methods when a precisely shaped transfer function is desired. For these reasons, one expects that programmable logic devices (PLDs) with high processing speed and low latency will prove to be invaluable as quantum and classical controllers.

PLDs are already a standard tool in industry and some areas of science, but they have yet to attain widespread use in fields such as quantum optics and quantum information science. Our aim in this paper will be to convey a base level of knowledge required to use these devices in representative experimental setups. First we promote the use of programmable logic with some potential applications. We then describe the details of practical implementation, from determining the required hardware specifications to completing the design flow. Finally we demonstrate this process with a familiar example of classical control by using a field-programmable gate array (FPGA) to stabilize the output of a Fabry–Perot cavity.

2. APPLICATIONS

The field of control theory has evolved tremendously since its inception. The generality of the formalism has led to a sophisticated set of rules and procedures applicable to control tasks in widely separated fields of science.[6–8] While we believe that complex control-theory-based strategies will play a major role in the broad quantum arena, here we wish to simplify the setting and justify the use of PLDs in terms of basic control objectives and physical constraints.

All control problems focus on a plant or a system that is the object of control. A typical plant has a set of input variables that can be driven and a set of output variables that can be measured. Plants can range in complexity from linear to nonlinear and from classical to necessarily quantum mechanical. Whatever the plant dynamics, all control goals can be quantified in terms of a metric which may involve both the input and output variables. The goal of the controller is to optimize this metric with an appropriate mapping from output variables to input variables.

Given infinitely fast and powerful detectors, controllers, and actuators, most control tasks are trivial. Practically, however, all components have limited bandwidth, and above certain frequencies any real controller will be effectively turned off. The relatively difficult part of designing most controllers is dealing with this physical reality. For example, most classical linear controllers must be designed with an appropriate phase response such that the system is not accidentally driven into unstable oscillations as the response magnitude necessarily descends below unity.[9]

The ideal controller should be fast enough to avoid being the rate-limiting step in any particular control loop. The outstanding feature of PLDs is that they can implement complex nonlinear logic at a high bandwidth and a low latency. Here latency refers to the delay between the time that a signal is received as input and the time that a calculation based on it becomes available as output. This reaction time is of little consequence in many data processing applications, but is critical in control loops. Essentially, the control bandwidth of any servo is limited by the inverse of this delay.

In many typical classical control situations PLD controllers are unnecessarily complex, and traditional analog circuitry would be adequate. For fundamentally quantum-mechanical experimental systems, the time scales are usually much smaller and the necessary algorithms much more complicated (such as the estimation of an operator expectation value). Thus using PLDs to orchestrate quantum control seems a natural and necessary marriage. In addition most types of PLD can be completely reprogrammed in a matter of minutes, allowing for a high degree of design flexibility in experimental situations. We now summarize a few potential classical and quantum scenarios in which PLDs clearly distinguish themselves from the alternatives.

A. Precise Linear Servos

In linear control tasks, PLD controllers have a distinct practical advantage over analog circuitry with regard to precision and flexibility. For example it is a well-known control problem to stabilize a plant over one of its resonances. An appropriate controller should precisely compensate for the measured center frequency and quality factor of the resonance. When creating an analog servo the designer must work with discrete components (resistors, capacitors, etc.) whose impedances have a nonnegligible error range. However a PLD transfer function can be specified digitally, making it much easier to match the system dynamics closely.

Figure 1 shows the near-compensation of a harmonic-oscillator (HO) resonance with a PLD antiharmonic-oscillator (AHO) transfer function. (Actually both transfer functions in the graph are implemented with a PLD by techniques described later.) Ideally the HO transfer function will be transformed into an integrator transfer function (with a constant $-90°$ of phase) when multiplied by the AHO compensator. The deviation from a perfect integrator is due to a slight error in the assumed damping. Refinements to the AHO design could remove this slight departure from the ideal.

PLDs will obviously not replace most linear servos in the typical laboratory, but the ability to optimize the stability of critical systems is a considerable resource. In Section 4 we will, for the purposes of demonstration, describe the use of a PLD controller to perform optimally a critical linear control task of major importance in the optics community: stabilizing a Fabry–Perot cavity.

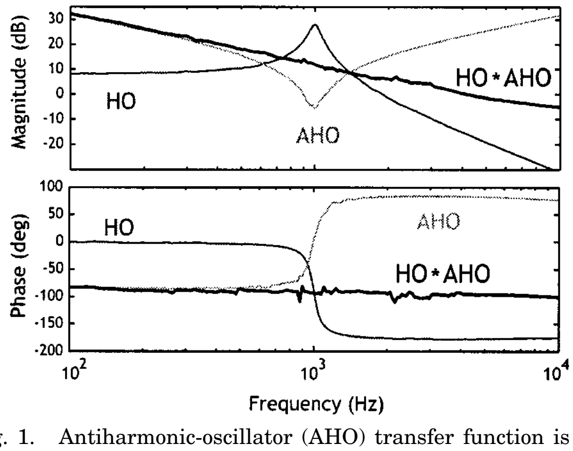


Fig. 1. Antiharmonic-oscillator (AHO) transfer function is designed such that its product with the harmonic-oscillator (HO) transfer function equals that of an integrator with a constant $-90°$ phase.

B. Optimal Measurement

In quantum feedback scenarios, either the measurement operators or the system Hamiltonian can be modulated in real time according to the information gained from a continuous measurement record. Consider the case in which only the measurement operators are adjusted. The goal of the entire measurement may be to determine most accurately the initial state of the system. Other situations may call for the measurement of only a single state parameter when all other state variables are either assumed or neglected. The authors have recently developed a system of this type for which the goal was to optimally measure the phase of a single pulse of light.[10] We constrain ourselves to measuring pulses that are not only long enough that their phase is well-defined, but long enough to allow us to feed back the measurement signal multiple times before the pulse has been completely destroyed by the detectors.

Wiseman and Killip have determined near-optimal measurement schemes for such a system based on quantum trajectory theory.[5] In short they consider the signal to be measured in an adaptive homodyne setup in which the pulse is mixed with a strong local oscillator whose phase Φ is continuously adjusted (within the duration of each pulse) according to the measured homodyne current I. To first order, the object of the algorithm is to lock to the side of the interference fringe; thus Φ is adjusted until I is zero.

Despite this simplistic description, the general optimal algorithm $(f: I \Rightarrow \Phi)$ is a highly nonlinear function based on estimation of state. It has been shown that the estimated state at any time is a function, remarkably, of only two parameters and the initial conditions. In terms of a scaled time v these parameters are

$$A_v = \int_0^v I(u)\exp[i\Phi(u)]\mathrm{d}u, \tag{1}$$

$$B_v = -\int_0^v \exp[2i\Phi(u)]\mathrm{d}u. \tag{2}$$

The phase of the local oscillator is usually taken to be $\Phi(v) = \hat{\phi}(v) + \pi/2$ where $\hat{\phi}(v)$ is the phase estimate to be used during the course of feedback. If one were to stop the feedback at any time, the best phase estimate would be $\hat{\phi}_C(v) = \arg(C_v)$ where $C_v = A_v v + B_v A_v^*$. However for subtle reasons $\hat{\phi}_C(v)$ should not be used as the estimate during the course of the feedback.

One simple algorithm uses $\hat{\phi}(v) = \arg(A_v)$. With this choice the algorithm simply reduces to a gain-scheduled integrator of the form

$$\mathrm{d}\Phi(v) = I(v)/\sqrt{v}, \tag{3}$$

where v is the time since the beginning of the pulse and the $v^{1/2}$ factor represents the effective gain. Currently this algorithm is being implemented with an FPGA that creates the $v^{1/2}$ gain factor with a lookup table representation of the function as described in Subsection 3.B.1.

More sophisticated algorithms (with optimal performance for certain squeezed states) have been proposed that use feedback of the form

$$\hat{\phi}(v) = \arg(C_v^{1-\epsilon(v)} A_v^{\epsilon(v)}), \tag{4}$$

where $\epsilon(v)$ is also a function of A_v and B_v. In this case the algorithm is sufficiently complex that any analog implementation would be extremely difficult to design.

In any case the nonlinear, low-latency behavior of PLDs suggest that they are a suitable tool for this task. Given that the form of a desired algorithm may change frequently with the introduction of realistic experimental complications, the rapid prototyping allowed by a PLD is also extremely convenient.

C. Feedback Control

When the goal is control rather than optimal measurement, a nontrivial Hamiltonian of the system will be controlled by the measurement record. Consider the case of an atom drifting through the light field of a small Fabry–Perot cavity. As has been demonstrated, the position of the atom may be imprinted onto the output light of the cavity.[3] This information can potentially be mapped back onto the intensity and phase of the input laser with the goal of trapping the atom in the cavity for extended periods of time.[11]

Optimal control of the atom's position will require a complex predictor–corrector structure in the feedback loop at microsecond time scales. If the associated calculations can be sufficiently reduced, a PLD with effective clocking speeds above $\sim$10 MHz will be able to perform this task. Of course the effectiveness of the control algorithm will depend on the assumed dynamics of the system from which it is derived. If the system needs to be described quantum-mechanically, we should institute a conditional quantum state estimator. If a classical description is sufficient, we can use a less complicated algorithm. The performance of different controllers will be a strong indicator of the validity of our descriptions. The ability to quickly redesign the PLD will be particularly advantageous when exploring this boundary.

Hamiltonian feedback can also be used to manipulate the internal states of atomic and molecular systems. Numerous groups have become interested in shaping femtosecond laser pulses to drive transitions which may be inaccessible using traditional means.[12] This includes the ability to synthesize rare molecular compounds. For example by iteratively reading the fluorescence spectrum of the system and intelligently moving in the parameter space of the pulse shape, one attempts to land at a shape conducive to creating the desired state or compound.

This procedure can happen in two regimes, learning control or feedback control. For learning control we consider using a new sample for every pulse, whereas for feedback control we consider using the same sample on every pulse. In the latter case the algorithm assumes that the sample has a long enough dephasing time (memory) that a significant degree of coherence is retained between pulses. For either case, a PLD-based controller may have significant advantages over alternative controller architectures.

D. Decision and Control for Quantum-Information Processing

In a generic quantum-computing architecture, there exist classical logic steps which involve performing a coherent quantum operation conditioned on the result of a measurement. For example quantum error-correcting codes can combat decoherence by mapping measured errors to appropriate correction operators.[13] In an experiment, this measurement-operation procedure should be performed much faster than the dephasing rate of the system. If the operations can be performed quickly upon command, PLDs will be able to orchestrate these codes in a reliable and reconfigurable fashion with minimal delay.

Even for nonconditional algorithms PLDs can streamline the implementation of complex instruction sets. In particular, groups working on ion-trap computing have developed means of performing entanglement algorithms,[1] but with an extensive overhead of macroscopic equipment that requires detailed manual adjustment whenever the algorithm is changed. Without pushing its computational limits, a PLD can be made to streamline such logic networks. By using software-defined algorithms, the users eliminate the time and risk of error associated with manual realignment of network components. Commercial magnetic resonance systems use PLDs for similar reasons.

As quantum-computing architectures grow to the point where conditional and nonconditional algorithms must be integrated in a way that is fast and flexible, programmable logic will be able to handle the task in a convenient manner. The success of any PLD controller will depend on its dynamic range and effective bandwidth. Next we discuss in more practical terms what levels of system performance can be reasonably expected from currently available PLDs.

3. DESIGN

A. Hardware

Once it is determined that a control algorithm needs to be implemented digitally, a designer is confronted with a wide array of possible controllers and corresponding acronyms. In addition to PLDs the options include conventional microprocessor systems, digital signal processors (DSPs) and application-specific integrated circuits (ASICs). Of course the choice of controller is highly dependent on the algorithm being implemented, because each device has its own trade-offs. Microprocessor systems are general enough to allow for a simple means of programming complex algorithms. However, these systems rely on a single-bus architecture which forms a significant bottleneck in signal processing applications. Overall throughput may be high but a large delay limits typical controllers to slow applications with kilohertz-scale bandwidths. In addition unreliable operating systems may present undesirable interrupt signals during critical stages of processing.

DSPs are specialized microprocessor systems with a multiple-bus design that are optimized for signal processing applications. Because of their parallel architecture DSPs can attain low-latency performance, but require a significant degree of sophisticated design expertise. ASICs are like PLDs in that the user designs them from the gate level, but ASICs are irreversibly hardwired for a single application. While PLDs generally have fewer resources available than ASICs, they offer an efficient parallel computation structure along with reprogrammability and a relatively simple design process.[14]

The market for PLDs is currently dominated by two companies: Xilinx Inc., and Altera Corporation. Devices from both companies have had extensive product development in industry, thus a substantial support network is available to designers. In choosing between PLD companies, several factors beyond the chip performance need to be considered, including the quality of the associated software environments. To obtain the maximum control bandwidth, we chose to work with an FPGA from Xilinx.

The logic structure of a Xilinx FPGA is designed to handle arbitrary algorithm architectures. The FPGA consists mostly of a grid with thousands of configurable logic blocks with programmable interconnections. Each configurable logic block contains a few small lookup tables which can serve as simple logic elements (AND, OR, etc.) when programmed. Also interspersed in this grid are blocks of RAM that can be programmed as user-defined functions with a large domain and range. Since each logic element needs to be triggered to operate, the distribution of a uniform clock signal with constant frequency and phase is a considerable design issue. Thus FPGA architectures commonly have digital clock managers or delay-locked loops that de-skew the clock signal across the device.

The performance of FPGA architectures has been increasing impressively in recent years. To give an indication of their current level of performance, we quote some of the characteristics of one of the top-of-the-line devices available on the market today. The Xilinx Virtex II can contain up to ten million system gates and have an internal clock frequency (f_C) of up to 420 MHz. The input–output speed can be above 840 megabauds per second which roughly matches the maximum speed of the best analog-to-digital converter (ADC) (100 million samples per second for a 12-bit sample Analog Devices, Inc., AD9432). This same FPGA has up to 192 SelectRAM™ blocks of 18 kilobits each. Because strong demand from industry drives the development of FPGA technology, these performance specifications will likely improve significantly in the near future.

Of course these devices must be coupled to a board, introducing some practical issues. The system used in the cavity lock described in Section 4 is a GVA-290 board (G.V. & Associates) with two Xilinx Virtex-E XCV1000E FPGA chips. Signals enter and exit the board through four input and four output SMA (subminiature version A) connectors. The signals are digitized by an ADC (Analog AD9432) at the input and converted back to analog by a digital-to-analog converter (DAC) (Analog AD9762) at the output. Each ADC is located on a detachable daughter board, allowing for converter upgrades and the addition of customized components and filters. Both the ADCs and DACs have 12-bit resolution and are driven at the clock speed of 100 MHz. A crystal oscillator provides the clock signal to the FPGA, which distributes a synchronized signal internally with delay-locked loops and also outputs the driving signal for the ADC and DAC at a controlled phase. Unlike standard models, the board was ordered with dc-coupled inputs, allowing us to have broadband control to dc. Boards often come with antialiasing analog filters, but these were not included here because of the substantial group delay a high-order filter can impose on the signal. The cost of this particular board including devices is approximately $10,000, but it should be stressed that functional systems could be assembled at far less cost. Xilinx also offers a special academic program through which university researchers can obtain the necessary software environment and a limited range of hardware products.

We can now discuss the latency and throughput of our controller in more detail. The latency is defined as the amount of time for an algorithm to process a single sample. The throughput is defined as the number of samples (or bits) per second of output from the device. For example consider a system of N components in series, each with the same sampling rate $f = 1/\tau$. Also assume the system is pipelined, meaning that a new sample is loaded every τ seconds and samples are registered (values held) between components. In this case the latency is $N\tau$, while the throughput is f. If this were a controller, the bandwidth of control would be limited to the inverse of the latency ($1/N\tau$), not the throughput.

One of the principal advantages of FPGA technology is that the delay can be quite small. Consider the case in which the FPGA of the GVA-290 board is programmed to pass a signal through without any manipulation. Figure 2 shows the transfer function and delay of this configuration. The ADC, FPGA, and DAC are all clocked at 100 MHz and each one takes a certain number of cycles (10 ns/cycle) to perform its function. The ADC imposes a delay of ten cycles, the buffers of the FPGA impose a delay of four cycles, and the DAC delays the signal only about one cycle. Adding all this, together with a small delay from other components, we find that below the Nyquist

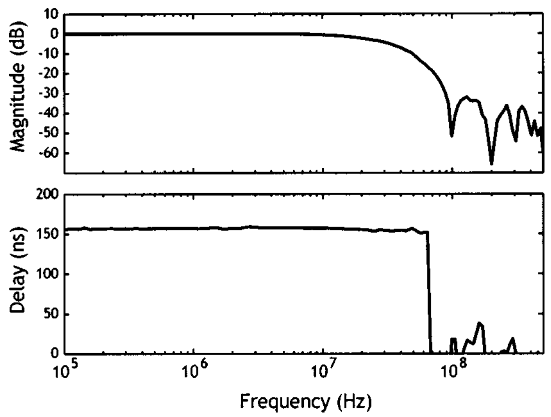


Fig. 2. Amplitude response and delay of the entire GV-290 board (ADC→FPGA→DAC). Note that the delay below the Nyquist frequency ($f_C/2 = 50$ MHz) is ~160 ns. The phase response in the constant delay region is linear with slope proportional to the delay.

frequency ($f_C/2 = 50\,\mathrm{MHz}$) the signal passes through at unity gain with a constant overall delay of ~160 ns. Thus the maximum control bandwidth for this device is ~6 MHz. Bandwidths in the tens of MHz may be anticipated with newer versions. If the FPGA algorithm is simple enough that the ADC dominates the delay, it may be desirable to use flash ADCs that have less latency at the expense of a larger power consumption and smaller number of output bits.

If the FPGA performs a complex calculation that requires multiple logical steps in series, the delay is increased by an integral number of cycles and the effective bandwidth suffers. A typical example is that of the finite impulse response (FIR) filter mentioned in Subsection 3.B.2 where, for B_U input bits, the sampling rate becomes f_C/B_U. For any general algorithm care should be taken to minimize the number of serial elements before implementation. If possible, calculations should be performed in parallel and lookup tables should be used to evaluate complicated functions.

B. Software

The design process for a particular algorithm has been largely automated with implementation-software environments such as Foundation™ ISE (integrated software environment, Xilinx). Once the design is entered through one of the options described below, the program steps through a series of compilation tasks before downloading onto the device. In order, the design is analyzed for syntactic errors, synthesized into a generic circuit, and implemented into an optimal bit stream appropriate to the particular device and board. The bit stream is then downloaded onto the device to achieve a stand-alone realization of the desired algorithm. Simulation programs are available at intermediate stages for debugging purposes. The latest version of Foundation ISE (4.1) compiles up to 100,000 gates/min. For reasonable designs an entire design flow can be expected to take ~ ten minutes. This allows for a rapid prototyping cycle, which is one of the most desirable features of this technology.

Numerous algorithm entry options are available. Using a library of primitive components, one can create a schematic of the desired circuit. Abstract finite state machine diagrams can also be interpreted. The third option is a text-based design written in either Verilog or VHDL [VHSIC (very high speed integrated circuit) hardware design language].

As is common in technology standards, the choice between Verilog and VHDL has become almost a matter of religion for everyday practitioners. It is worth pointing out some of the accepted differences between the languages. Verilog is generally regarded as being easier to learn. A large majority of engineers implementing commercial systems use Verilog. Historically VHDL was meant as a description language before being adopted as a means of synthesis. As a result, VHDL is a much more strongly 'typed' language. The range of abstraction is also different between the two languages. Although there is a considerable overlap, Verilog extends to a lower level of abstraction while VHDL extends to a slightly higher level. For noncritical reasons we chose to design in VHDL, so we will discuss the following designs in those terms. However the discussion is abstract enough that most concepts apply to both languages.

To first order, VHDL is a text-based description of a schematic design. The mapping between input and output bus variables consists of a series of abstractly defined components in which output ports are connected to input ports with defined signal variables. Each component has an associated entity and architecture, where an architecture is an instantiation of an entity. For example a component with entity 'op-amp' (with only input and output ports defined) could have its functionality determined by the particular architecture 'op27.' The internal workings of a particular architecture can be specified in another VHDL file with more components that are defined elsewhere. In this way the code lends itself nicely to nested level of detail and organized project design. Also one can easily swap out components by changing architectures, but not entities, within the code.

At some point in the hierarchy, primitive components must be called upon. The Xilinx software offers an extensive library of such components (AND, OR, etc.) for use with each particular device. In addition to these basic primitives one can also create more complicated, but commonly used, components with the Xilinx core generator. These objects (adders, multipliers, filters, DSP elements) can be customized with user-specified parameters.

Each component loads inputs and returns outputs triggered by an input clock signal. Hence when designing in VHDL one thinks in terms of circuit diagrams in which, on every clock cycle, events happen concurrently across the device. On the other hand in traditional C-like computer languages events progress in a serial manner. At times serial logic is convenient and in fact VHDL offers a restricted form of serial logic in a form known as a process. These processes are bits of C-like code that execute when triggered. Inside a process, variables can be manipulated with functions defined in other VHDL files. However a signal can be changed only once within a process. For this and other reasons, processes are best used as referees to generate secondary triggering signals and logic. While processes can perform some level of math, the very difficult math is best left to the components which have been streamlined for such purposes.

An appropriate use of a process is to initialize parameters and control timing. For example Fig. 3 demonstrates how the simple adaptive phase algorithm mentioned above is implemented. Both the VHDL and an equivalent schematic are shown. The photocurrent I enters the device and is multiplied by the time-dependent gain factor $G(t) = 1/\sqrt{t}$, which is created by sending the time signal t through a lookup table (described in Subsection 3.B.1). The resulting signal $\mathrm{d}\Phi(t) = I(t)/\sqrt{t}$ is then sent to one port of an adder with the other input port being wired to the output signal $\Phi(t)$. Because the output is connected to the input with a delay, the adder serves as an integrator and executes the relation $\Phi(t) = \Phi(t-1) + \mathrm{d}\Phi(t)$ at every time step. The process plays an important role in this algorithm by initializing the integral value and creating the time signal. At the beginning of the pulse (integration), the process initializes t and Φ to zero. At each subsequent clock signal the process increments t by one and lets the adder integrate the signal.

At the end of the pulse, the process waits for the next pulse then repeats the sequence. Figure 4 shows the algorithm in action. Through the integrator structure, Φ is adjusted until I is locked to zero. The overshoot is a result of the FPGA delay.

A single measurement using this algorithm is shown in Fig. 4. Here the pulse is a 50-μs slice of a weak cw coherent beam. The feedback algorithm is sampling at 100 MHz with a delay of less than 1 μs. Because of the delay and other bandwidth limiting components in the loop, our effective feedback bandwidth is limited to ~1 MHz.

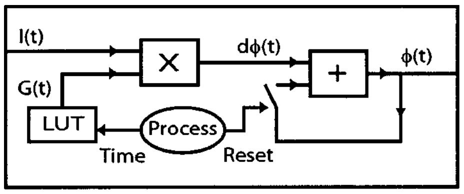
As will be demonstrated in Section 4, MATLAB® plays a complementary role in the design process. It can be used to create the necessary coefficients and memory blocks used as parameters in the VHDL components. In particular the control and DSP toolboxes provide relevant functionality. Also Simulink® is a good tool for simulating the associated experiments, where delays and other realistic factors can complicate the dynamics. There are other software packages that attempt to translate directly from a Simulink design of an algorithm into equivalent VHDL, but they are in early stages of development.

Because of their extensive utility, RAM lookup tables and filter components are worth discussing in greater detail.

1. *Lookup Tables*

Most FPGA chips come equipped with large blocks of internal RAM that can be used as generalized functions or lookup tables (LUT). Given an amount of memory on a particular block, the user can decide on a certain number of input and output bits. During operation the RAM block returns the value held at the address specified by the input, effectively implementing the desired function. For example, on the XCV1000E, 160 blocks of 4096 $= 2^{12}$ bits are available for internal use. (As noted above, the Virtex II devices have much larger 18-kilobit blocks.) To make one block behave as the function f with B_i input bits, the designer would choose the output to be $B_o = 2^{12-B_i}$ bits. Possible partitions are $(B_i, B_o) \in [(1, 2048), (2, 1024), (3, 512), \ldots, (8, 16), \ldots, (12, 1)]$. Once a partition is chosen, the designer would use MATLAB to define a block of data consisting of 2^{B_i} values each of size B_o bits and use this block of data as a parameter in the VHDL LUT component. If the discretization is a problem, more RAM blocks can be used to represent the function. If desired the memory of a RAM block can also be written dynamically during operation. With this ability, an algorithm could easily adapt itself according to the signals it receives. Both the read and write operations (from and to one RAM address) take only a single clock cycle.

As mentioned above, these LUT functions play an extremely important role in speeding the functionality of nonlinear algorithms. The application may be as simple as nonlinear gain scheduling of a controller, or as complicated as full quantum-mechanical state estimation with the LUT performing functions based on assumed system parameters. In general it is a matter of judgment how to partition complex algorithms, but any optimal partition will likely involve the use of LUTs to perform the difficult parts of the calculation with minimal time delay.



VHDL Equivalent (the symbol -- precedes comments)

```
 --first component is the look-up table
 --component format is 'instance: type'
 --port map plugs signals into component ports;  _# is label for bit size of bus

 lut_num1 : ramblock_core
   port map (EN=>vcc_sig, WE=>gnd_sig, RST=>gnd_sig, CLK=>clksys,
     ADDR=>time_8,DO=>Gtime_16,DI=>Gtime_16);

 multiplier_num1 : multiplier_core
   port map (A => I_12, B => Gtime_16, CLK => clksys, P => dphi_28);

 --trim signal back down to size
 dphi_12 <= dphi_28(27 downto 16);

 adder_num1 : adder_core
   port map (A => dphi_12, B => phi_21_a, Q => phi_21_b, CLK => clksys);

 --plug signals together
 phi_21_c <= phi_21_b;

 --start process on clock change
 PROCESS(clksys)
 VARIABLE time : integer;
 BEGIN
   --trigger on rising edge of clock
   IF  clksys='1' AND clksys'EVENT THEN
    IF time < tau_experiment THEN
      phi_21_a <= phi_21_c;
      phi_12 <= phi_21_c(20 downto 9);
    ELSE
      --zero signals during dead time
      phi_21_a <= "000000000000000000000";
      phi_12 <= "000000000000";
    END IF;
    IF time = tau_experiment+tau_dead THEN
     time := 0;
    END IF;
    time := time+1;
    --convert variable to signal
    time_8 <= int_to_bus(time);
   END IF;
 END PROCESS;
```

Fig. 3. FPGA schematic and corresponding code for the adaptive phase measurement algorithm. In the schematic the process is not represented as a block component because it is coded in a serial manner.

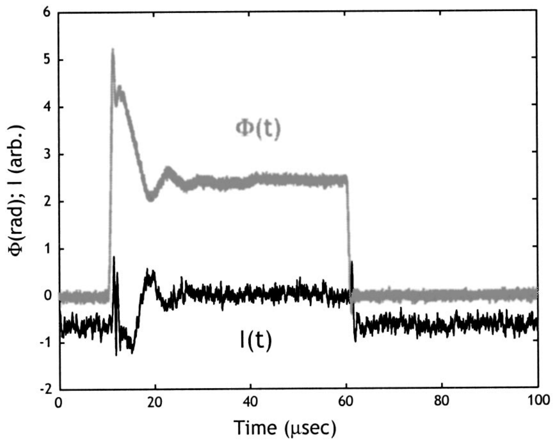


Fig. 4. $\Phi(t)$ and $I(t)$ trajectories for the phase measurement of a single pulse of light. The current is locked to zero and the ending point of the phase is a rough estimate of the measured phase. The true phase measurement is a functional of both traces. The oscillations are due to the delay in the loop.

2. *Filters*

PLDs have a clear edge over analog circuitry in nonlinear processing, but they also have a potential advantage in implementing precise, generic linear filters and transfer functions. A standard core element offered by Xilinx is the FIR (finite impulse response) filter. The FIR filter is defined in discrete time as

$$y(n) = \sum_{i=0}^{N} a(i)u(n-i), \qquad (5)$$

where $y(n)$ and $u(n)$ are the output and input, respectively, at the discrete time n. With standard MATLAB functions (**firls**, **remez**) one can specify an arbitrary amplitude response and get out the corresponding $a(i)$ vector. The sampling frequency for a FIR element is $f_F = f_C/B_U = 1/\tau_F$ where B_U is the number of bits chosen to represent $u(n)$. Of course the filter is useless at shaping the response above this frequency. The group delay of the signal through the filter is approximately $\tau_F N/2$.

The range of attenuation is also a concern in the design of any filter. For an FPGA with B_F bits entering and leaving, the dynamic range is $20\log(2^{B_F})\,\mathrm{dB}$. For our board with 12-bit ADC and DAC inputs and outputs, this corresponds to 70 dB. The designer should also have a sense of the size of the input and output signals. If the input signal is too high, the FPGA will rail; if the input is too low, it will fail to rise above the smallest bit size. To avoid problems of this kind, broadband gain elements can be used at the input and output of the FPGA board.

A drawback of the FIR design is that it cannot be used to control the phase response of its transfer function. On the other hand, a generic continuous-time linear transfer function

$$G_C(s) = \frac{c(N)s^N + c(N-1)s^{N-1} + \ldots + c(1)}{d(N)s^N + d(N-1)s^{N-1} + \ldots + d(1)}, \qquad (6)$$

where $Y_C = G_C U_C$ has phase control built in through the denominator. To approximate this function on a PLD requires an infinite impulse response (IIR) filter.

One possible IIR design process illustrates this need. To generate a digital IIR design, first create $G_C(s)$ using standard control techniques (Nyquist, linear quadratic regulator, etc.). Next convert from a continuous to a discrete transfer function

$$G_C \Rightarrow G_D(z) = \frac{a(0) + a(1)z^{-1} + \ldots + a(N)z^{-N}}{b(0) + b(1)z^{-1} + \ldots + b(N)z^{-N}} \qquad (7)$$

with the MATLAB function **c2d**. We have used the definition $Y_D = G_D U_D$ in the discrete time representation. Apply a z-transform ($z^{-1} \Rightarrow$ unit delay) to create the discrete time difference equation

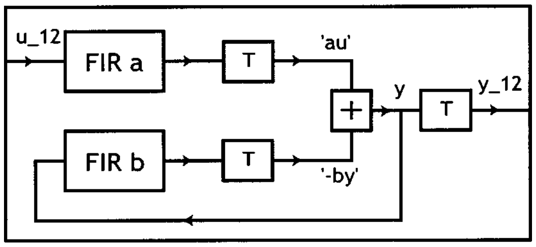


Fig. 5. Implementation of an IIR filter. T components trim a certain number of least-significant bits from the data bus.

$$y(n) = \sum_{i=0}^{N} a(i)u(n-i) - \sum_{i=1}^{N} b(i)y(n-i), \qquad (8)$$

with the definition $b(0) = 1$. Finally implement the difference equation in hardware as in Fig. 5 with two FIR blocks and one adder.

With $b(n > 0) = 0$ the filter is just a FIR filter. However with $b(n > 0) \neq 0$ the output is fed back to itself. Hence an impulse response will have an infinite effect on the output. Of course with internal feedback loops, the system is potentially unstable to noise and rounding errors. For this reason, among others, the Xilinx core generator does not create flexible IIR modules.

However with careful consideration of the number of bits required at each stage, a stable IIR filter can be created as in Fig. 5. The sampling frequency for this simple architecture is $f_C/2B_Y$ where B_Y is the number of bits used to keep track of $y(n)$ internally. The factor of two results from the delay of both the adder and the FIR element. Because of the feedback, the IIR filter can achieve a given amplitude response with a lower number of coefficients than the FIR filter. This means the filter delays the signal less. Even though the IIR has fewer coefficients than an analagous FIR filter, the coefficients of the IIR filter have to be specified to a greater degree of precision to achieve the same amplitude response.

4. SPECIFIC EXAMPLE: CAVITY LOCK

We now discuss the use of an FPGA to perform a classical task necessary for low-noise experiments. High-precision optical measurements demand laser intensity noise be minimized as much as possible. In the adaptive phase experiment mentioned above, the input laser is a Lightwave Nd:YAG Model 126 (1064 nm) with an inherent broad relaxation-oscillation noise peak at ~500 kHz. To perform broadband detection and control near 1 MHz, this intensity noise must be removed from the beam with a Fabry–Perot cavity. In this section we will detail the use of an FPGA to control the output of this filtering cavity. We consider this approach to be novel and effective (albeit expensive). However our main interest is in illustrating the use of this device, which is unfamiliar to many in experimental optics, to perform a task which is very familiar.

A block diagram of the system is shown in Fig. 6. The output intensity of the cavity is stabilized using the standard Pound–Drever–Hall method in which the error signal, which depends on the laser-cavity difference frequency, is created from a reflected carrier beam with sidebands. At frequencies below 100 Hz the feedback loop is dominated by a piezoelectric element (lead zirconate titanate, PZT) which controls the length of the cavity. At higher frequencies and through the closing point of the servo, the feedback is from an AOM (acousto-optic modulator) driven by a VCO (voltage-controlled oscillator) which adjusts the frequency of the input beam.

Given the control architecture of Fig. 6, the design process can be made very systematic with the flexibility of the FPGA. Because the critical behavior of the servo will be dominated by the VCO–AOM loop, we concentrate on

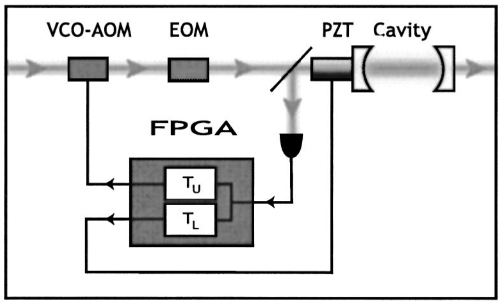


Fig. 6. Feedback architecture for a Fabry–Perot Cavity. The electro-optic modulator (EOM) puts sidebands on the beam necessary to generate the locking signal. The FPGA algorithm T_U (upper) maps the error signal to the fast VCO–AOM frequency shifting combination. The FPGA algorithm T_L (lower) maps the signal to the slow PZT.

the design of T_U (upper). First the transfer functions of the elements in the loop are measured. Here we find that the VCO–AOM combination behaves like a low-pass filter (T_V) with a corner at 100 kHz. The cavity itself can be modeled as a low-pass filter (T_C) with a corner at about 10 kHz (the cavity linewidth). The goal is to design T_U such that the closed-loop transfer function $T_{CL} = T_C T_V/(1 + T_C T_V T_U)$ is stable.

At this point, we can use the Matlab control toolbox to design an optimal T_U. One option is to provide the function **lqr** with the state space representations of T_V and T_C and an appropriate cost function to create the optimal T_U. The result simply tells us to make the combination $T_C T_V T_U$ behave like an integrator ($T_1 = 1/s = 1/j\omega$) such that the controller satisfies the Nyquist criterion with 90° of phase margin.

There are practical problems with this approach. In particular the optimal T_U has unrealistic infinite gain at very low and very high frequencies. Fortunately we can approximate the ideal case without compromising the control characteristics by flattening the response of T_U below 100 Hz (where the PZT arm takes over) and rolling off the response at 300 kHz, beyond the closing point of the servo. So instead of making $T_U = T_1/T_C T_V$ we use $T_U = T_{LP1} T_{LP2}^2 / T_C T_V$ where T_{LP1} is a low-pass filter with the corner at 100 Hz and T_{LP2} is a low-pass filter with the corner at 300 kHz.

To get high gain at frequencies below 100 Hz, we make T_L (lower) behave as a low-pass filter with a corner at only a few Hz. A better choice would be to implement T_L as a high-gain analog integrator, but we use the FPGA to implement T_L here for purposes of demonstration.

Next we generate proper IIR coefficients for both paths by the method described previously, treating T_L and T_U as the continuous transfer function G_C. With a clock frequency of 100 MHz and an internal sample size of $B_Y = 32$ bits, the IIR structure has an effective bandwidth of 1.5 MHz ($f_C/2B_Y$), which is adequate to generate the critical features of the transfer function.

Figures 7 and 8 show the desired and actual transfer functions of both arms. Each arm fails to match the desired phase and amplitude response in a similar way. First because of the finite size of the sampling time, the actual phase response differs from the desired response as the frequency approaches the effective sampling frequency. In fact this mismatch happens below the sampling frequency because of the delay of the IIR filter. Second at low frequencies, the FPGA gives less gain than desired. This is because we are dealing with finite precision coefficients. The price paid for having a large sampling frequency with small delay is that we have less control over the size of the low-frequency gain. Finally note that the PZT arm integrator achieves the full 70 dB of expected range (input and output size is 12 bits).

The closed-loop transfer function behavior for both arms matches our expectations for noise rejection at low frequencies. A mismatch at higher frequencies is due to inadequate modeling of the PZT and other components. (The PZT, modeled as a simple low-pass filter, behaves more like a collection of oscillators with different resonances.) Qualitatively the FPGA lock was much more robust with respect to high-frequency noise than an analog version of the servo. This was likely due to the precise match to the plant dynamics near the unity gain point of the servo achieved by the use of large FIR coefficients. However the FPGA was unable to retain the lock over time scales of more than a few hours because of the saturated gain at very low frequencies. This problem could easily be remedied by using an analog integrator with more dc gain to replace the FPGA PZT transfer function. The main advantage of the FPGA is its fast, accurate re-

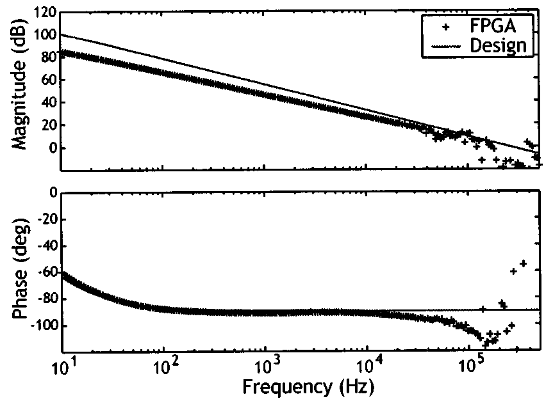


Fig. 7. Bode plot of T_L (lower transfer function leading to PZT). The design is a low-pass filter which dominates control below ~100 Hz.

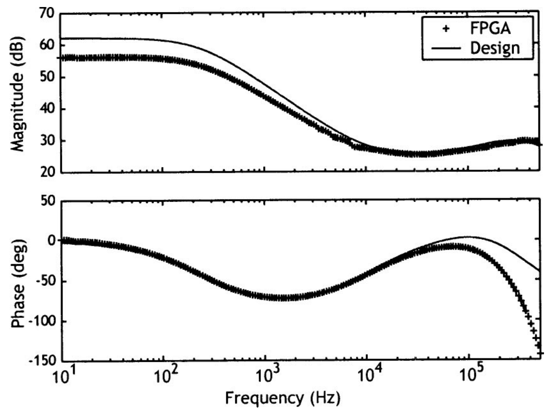


Fig. 8. Bode plot of T_U (upper transfer function leading to VCO–AOM). The peak in phase is designed to stabilize the plant through the unity gain point.

sponse and, aside from the demonstration presented here, there is no practical reason to use the FPGA for high-gain, low-frequency applications.

Finally another feature of FPGA control is the possibility of adding logical automation to this system. Specifically the FPGA could be programmed, in case the controller loses the lock, to sense this condition, sweep for a signal, zero in, and reacquire the lock. The abstract logical nature of VHDL code makes this task simple compared with the procedure needed to create an acquisition system using standard electronics.

5. SUMMARY

To demonstrate the use of programmable logic technology in an otherwise familiar setting, we have concentrated on a linear control application. We have used this example to convey the issues associated with a digital controller, including design, latency, and discretization. However we have only hinted at the more interesting advanced applications in experimental quantum optics which are sure to develop more quickly because of this technology. FPGAs and similar devices are particularly suited to any physical system where nonlinear mappings are desired between output and input variables within the natural dynamical time scale. With these devices and sufficiently protected quantum systems in hand, the field of coherent quantum control may soon have enough speed to match the intelligence of its proposed controllers.

ACKNOWLEDGMENTS

J. Stockton acknowledges the support of a Hertz Foundation Fellowship and H. Mabuchi acknowledges the support of an A. P. Sloan Research Fellowship. This work was supported by the National Science Foundation under grant PHY-9987541, and by the Office of Naval Research under Young Investigator Award N00014-00-1-0479.

J. Stockton may be reached by e-mail at jks@caltech.edu.

REFERENCES

1. C. A. Sackett, D. Kielpinski, B. E. King, C. Langer, V. Meyer, C. J. Myatt, M. Rowe, Q. A. Turchette, W. M. Itano, D. J. Wineland, and C. Monroe, "Experimental entanglement of four particles," Nature **404**, 256–259 (2000).
2. M. R. Andrews, M.-O. Mewes, N. J. van Druten, D. S. Durfee, D. M. Kurn, and W. Ketterle, "Direct, nondestructive observation of a Bose condensate," Science **273**, 84–87 (1996).
3. C. J. Hood, T. W. Lynn, A. C. Doherty, A. S. Parkins, and H. J. Kimble, "The atom-cavity microscope: single atoms bound in orbit by single photons," Science **287**, 1447–1453 (2000).
4. A. C. Doherty, S. Habib, K. Jacobs, H. Mabuchi, and S. M. Tan, "Quantum feedback control and classical control theory," Phys. Rev. A **62**, 012105 (2000).
5. H. M. Wiseman and R. B. Killip, "Adaptive single-shot phase measurements: the full quantum theory," Phys. Rev. A **57**, 2169–2185 (1998).
6. B. Rahn, A. C. Doherty, and H. Mabuchi, "Exact performance of concatenated quantum codes," Phys. Rev. A **66**, 032304 (2002).
7. A. C. Doherty, P. A. Parrilo, and F. M. Spedalieri, "Distinguishing separable and entangled states," Phys. Rev. Lett. **88**, 187904 (2002).
8. M. E. Cseta and J. C. Doyle, "Reverse engineering of biological complexity," Science **295**, 1664–1669 (2002).
9. O. L. R. Jacobs, *Introduction to Control Theory* (Oxford University, Oxford, UK, 1993).
10. M. A. Armen, J. K. Au, J. K. Stockton, A. C. Doherty, and H. Mabuchi, "Adaptive homodyne measurement of optical phase," Phys. Rev. Lett. **89**, 133602 (2002).
11. S. Habib, habib@lanl.gov, K. Jacobs, and H. Mabuchi are preparing a manuscript to be called "Feedback control of atomic motion in an optical cavity."
12. H. Rabitz, R. de Vivie-Riedle, M. Motzkus, and K. Kompa, "Whither the future of controlling quantum phenomena?," Science **288**, 824–828 (2000).
13. M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University, Cambridge, UK, 2000).
14. D. Stranneby, *Digital Signal Processing: DSP and Applications* (Newnes, Oxford, UK, 2001).