Matlab Data Acquisition and Control Toolbox for Basic Stamp Microcontrollers

Anshuman Panda[1], Hong Wong[2], Vikram Kapila[2], and Sang-Hoon Lee[2]

Abstract—In this paper, we present a Matlab and Simulink based software platform that enables the use of inexpensive microcontrollers for data acquisition and control tasks. The proposed framework is well suited for data acquisition and control tasks that require graphical user interface (GUI) and/or advanced computational capabilities but do not require stringent hardware performance. We illustrate the efficacy of our data acquisition and control technique by performing position control of a DC motor using a Basic Stamp 2 (BS2) microcontroller and our Matlab data acquisition and control toolbox.

I. INTRODUCTION

Data acquisition and control boards, also known as DAC boards, are used in virtually every field of engineering to establish communication between sensors/actuators and decision making devices, e.g., a PC. In academia these boards are used from entry level laboratory experiments for physics and chemistry to upper level laboratory experiments in automatic controls and signal processing.

A variety of PC-based DAC boards are available from vendors such as Advantech [1], National Instruments [2], and Quanser [3]. Furthermore, DAC solutions from several of these vendors support icon-based programming environments (e.g., Simulink [4] and LabVIEW [5]) for implementing measurement and control algorithms. Unfortunately, many PC-based DAC boards tend to be expensive. In a recent price comparison of several Matlab and Simulink supported PC-based DAC boards of different form factors (e.g., PCI, ISA, and PCMCIA), we determined that these DAC boards cost from $500 to several thousand dollars. Moreover, many of these DAC boards include an array of features (e.g., high sampling rates, high resolution analog to digital converters (A2Ds), etc.) that a typical user may not even utilize to the fullest potential.

The existing PC-based DAC boards supporting Simulink's icon-based programming environment require several additional software technologies. Specifically, in order to incorporate functionality of a particular Matlab-supported DAC board, the user needs Matlab's Real-Time Workshop (RTW) toolbox and a C compiler. In addition, the user *must* provide a real-time environment (e.g., Real-Time Windows Target for Matlab-based DAC solutions, Real-Time Extension for Quanser-based DAC solutions, etc.) for real-time execution of the designed algorithm. However, in some cases, the user may not require real-time execution of the algorithm (e.g., environmental monitoring). Thus, similar to PC-based DAC hardware solutions, the software requirements for the existing PC-based DAC solutions may be excessive.

In contrast to the PC-based DAC boards, microcontrollers are inexpensive devices (costing only a few tens of dollars), which are widely used for embedded computing. However, a majority of microcontrollers require programming using one or other embedded programming variants of high-level programming languages (e.g., Basic, C, Java, etc.). In addition, many low-cost microcontrollers do not allow floating-point numerical computations that may be needed to implement advanced feedback control algorithms.

In this paper, we present a Matlab and Simulink based software platform that enables the use of inexpensive microcontrollers for data acquisition and control tasks. Specifically, we develop a low-cost PC-based DAC board using Parallax Inc.'s BS2 microcontroller [6]. Furthermore, we provide a library of BS2 functions for Simulink. Next, we exploit Simulink's icon-based programming environment to implement user-defined algorithms in a block diagram format. In addition, we build upon the foundation of [7] to exploit Simulink and Matlab's built-in serial communication capabilities to communicate with various sensors and actuators connected to a BS2 microcontroller. The BS2 microcontroller was selected to illustrate our DAC toolbox since Matlab/Simulink device library for BS2 can be easily developed and implemented by exploiting BS2's Basic style instructions that are simpler *vis-à-vis* instruction sets of other microcontrollers. Moreover, our BS2 device library requires only minor modification for use of new types of sensors and actuators with our Matlab DAC toolbox.

The framework of this paper is significant for several reasons. First, by extending the capabilities of Simulink to low-cost microcontrollers, our method overcomes two limitations common to most microcontrollers, *viz.*, the lack of advanced icon-based software interface for efficient development of control algorithms and the lack of a GUI to

This work was supported in part by the National Science Foundation under an RET Site Grant 0227479 and a GK-12 Fellows Grant 0337668, and the NASA/NY Space Grant Consortium under Grant 39555-6519.

[1] Electrical and Computer Engineering Department, Polytechnic University, Brooklyn, NY, 11201.

[2] Mechanical, Aerospace, and Manufacturing Engineering Department, Polytechnic University, Brooklyn, NY, 11201.

allow intuitive interaction. Second, our DAC platform is very economical since it requires the use of only an off-the-shelf BS2 microcontroller (under $50) and obviates the need for overhead software such as RTW required for most PC-based DAC systems, thus further lowering the cost of acquiring such a system. This affords an opportunity to students to conduct industry-style rapid control prototyping and hardware in the loop experiments. Third, the ability to interface and program microcontrollers using the intuitive graphical programming environment of Simulink provides the flexibility and versatility of equipping a wide array of undergraduate-level laboratories (physics, measurement systems, feedback control, and mechatronics) at an economical cost using our DAC platform. Fourth, our microcontroller-based DAC system is inherently portable due to its small size and low power requirement, extending its benefits to students who can acquire their personal DAC system for capstone design projects and for experimental research.

II. Components Overview

Our Matlab based BS2 DAC system is composed of two main components, hardware and software. The hardware required for this DAC system is the BS2 microcontroller and user selectable sensors and actuators. The software required for this system is Matlab with serial communication capability and Simulink.

A. *Basic Stamp 2 Microcontroller*

The BS2 is a popular microcontroller used both in hobby and industrial projects. The BS2 has 16 general-purpose digital input/output (I/O) pins. Throughout the rest of this paper, we will refer to these digital I/O pins as pins. The high state on a BS2 pin refers to a 5VDC and a low state on a BS2 pin refers to a 0VDC (ground potential). See [6] for further details on the BS2 microcontroller hardware.

Additional hardware used in this paper is the Board of Education (BOE) development board. The BOE provides *i)* built-in circuitry for programming the BS2 microcontroller and serial data communication, *ii)* an interface for a power supply, *iii)* user accessibility of the BS2 pins, and *iv)* a breadboard area for custom circuits. In this paper, the BOE is an interface allowing for easy connectivity to sensors and actuators.

B. *DB-9 Serial Cable:*

Serial communication from the BS2 to a PC is performed through a DB-9 serial cable. This cable facilitates data communication between the BS2 and the PC and allows programming the BS2 from the PC. See [7] for further details on serial communication and DB-9 serial cable.

C. *Matlab*

Matlab is the primary software environment for our BS2 DAC toolbox. In addition to providing functionality for the Simulink toolbox, the Matlab scripting language allows for high level development of functions that can interface with specific hardware. We exploit this capability by using built-in serial communication functions provided by Matlab, versions 6.1 and higher.

D. *Simulink*

Simulink is MathWorks' icon-based programming environment that allows the user to construct a block diagram representation of a system. In a Simulink block diagram, the user can insert various blocks to construct or simulate system dynamics, control architectures, etc. The Simulink toolbox contains many libraries whose elements can be embedded in Simulink block diagrams. Simulink also provides user-defined blocks, in the form of s-function blocks, which can be modified to perform user-defined tasks. Furthermore, every Simulink block allows for a set of "callback functions," which execute upon specific events when running a Simulink block diagram. For example, in Figure 1, the "Simulation start" callback function, **TotalCompile**, will execute before the Start state of the Simulink block diagram. See [8] for further details on callback functions.

III. Software Interface

Referring to Figure 1 the software interface of our data acquisition and control system consists of two main components: *i)* a Simulink model file named *Template.mdl* and *ii)* a block library named BS2Library. *Template.mdl* is the file where the user designs the Simulink block diagram for interaction with the BS2. BS2Library is the library of additional blocks that communicate data with sensors and actuators connected to the BS2 microcontroller.

A. *Template.mdl*

The *Template.mdl* model file is an empty Simulink block diagram, where the user designs the Simulink block diagram for interaction with the BS2. The key property of *Template.mdl* is the inclusion of a function within the callback parameters of this Simulink model file, where this function is to be executed before the start of the block diagram. Furthermore, renaming this file still preserves this property, whereas opening a new Simulink model file does not.

When starting the Simulink block diagram, the callback function called **TotalCompile** is executed first. This function performs several important tasks and enables the communication between Matlab and the BS2 microcontroller. Details of this function are provided in a subsequent section.

B. *BS2Library*

The BS2Library is a custom library for Simulink, which provides blocks (in the form of s-functions) that interface with sensors and actuators connected to the BS2. See Table I for a complete listing of sensor and actuator blocks currently available. Furthermore, this library contains a block labeled

IOBlock that enables serial communication between the BS2 microcontroller and Matlab and computes the sampling period of a block diagram. This block is required in all user-designed Simulink block diagrams that will incorporate sensors and actuators connected with the BS2 microcontroller. Figure 2 provides a graphical description of the BS2 Library.

1) Sensors and Actuators: Sensor and actuator blocks provided in the BS2 library are used to communicate with sensors and actuators connected with the BS2 microcontroller. Each sensor or actuator block contains block parameters that need to be set for appropriate hardware configuration. The following describes hardware settings and parameter requirements of each block.

RCtime block: This block measures the time taken by a specific pin on the BS2 to change its state. It is used to obtain measurements from a variable resistance/capacitance sensor. The variable resistance/capacitance sensor is required to be connected in a series resistor-capacitor circuit with a constant capacitor/resistor, i.e., if the sensor is a variable resistor, a constant capacitor is required, and vice versa. See [9] for further details on how to construct an RC circuit. The RCtime block requires two parameters. The first parameter is the BS2 pin on which the BS2 monitors the state of the resistor-capacitor circuit. The second parameter is the initial state of the resistor-capacitor circuit.

AtoD_LTC1296 block: This block can receive voltage data from an LTC1296 A2D IC. The LTC1296 A2D IC, manufactured by Linear Technology Inc., is a 12-bit A2D (11-bit plus an additional sign bit) that has 8 single input channels, which can be used as 4 differential inputs, and requires a +/-5VDC power supply. Furthermore, this IC is controlled by the BS2 via the serial peripheral interface (SPI). The AtoD_LTC1296 block requires five parameters. The first parameter is the BS2 pin on which sensor data (in the form of a voltage signal) from the LTC1296 IC is sent to the BS2. The second parameter is a specific channel on the LTC1296 IC that contains the sensor data. The third parameter is the BS2 pin from which the BS2 sends the “clock” signal to the LTC1296 IC. The fourth parameter is the “chip select” pin and corresponds to the BS2 pin that enables operation of the LTC1296 IC. Finally, the fifth parameter is the BS2 pin from which the configuration information for the LTC1296 IC is sent. See [10] for further details on the LTC1296 IC.

ServoMotor block: This block controls standard servo motors connected to a BS2 microcontroller, one of which is the Parallax servo motor manufactured by Futaba Corp. The ServoMotor block requires one parameter, the BS2 pin which transmits a pulse-width modulated signal that controls the servo motor. See [11] for further details on how to operate a servo motor from the BS2.

DtoA_MAX537 block: This block sends voltage data supplied by the Simulink block diagram to the MAX537 D2A IC. The MAX537 D2A IC, manufactured by Dallas Semiconductor Inc., is a 12-bit D2A (11-bit plus an additional sign bit) that has 4 single output channels, which can be used as 2 differential outputs, and requires a +/-5VDC power supply. The MAX537 IC requires four parameters. The first parameter is the BS2 pin from which actuator data (in the form of voltage output) is to be sent to the MAX537 IC. The second parameter is a specific output channel on the MAX537 IC. The third parameter is the BS2 pin from which the BS2 sends the “clock” signal to the MAX537 IC. Finally, the fourth parameter is the “chip select” pin. See [12] for further details on the MAX537 IC.

Finally, sensor and actuator blocks are responsible for writing to or reading from global variables, which are to be sent or received from the BS2, respectively. Details of these operations are provided in a subsequent section.

2) IOBlock: The main purpose of this block is to *i)* initiate serial communication between the BS2 microcontroller and Matlab, *ii)* send and receive data between BS2 and Matlab, and *iii)* terminate this serial communication link.

The initiation and termination of serial communication is performed by two functions, **OpenSerialPortScript** and **CloseSerialPortScript**, which are executed at the Start and Stop state of the Simulink block diagram, respectively. For a set of sensors and actuators to be used in a Simulink block diagram, the **IOBlock** performs serial communication with the BS2 such that the order of data received from sensors and sent to actuators is determined by the **TotalCompile** function. Details of serial data communication are provided in a subsequent section.

The **IOBlock** is programmed to be the first block executed in the Simulink block diagram. This ensures that all sensor and actuator data in Matlab is first received and sent, respectively, which then is used by the appropriate sensor and actuator blocks in the Simulink block diagram.

Aside from serial communication, another important task of **IOBlock** is to provide the sampling period of a given Simulink block diagram. This feature allows blocks that require the sampling period to be used properly, e.g., an integrator block. We denote the sampling period as the amount of time required for one cycle of the Simulink block diagram to execute. The sampling period provided by **IOBlock** is an averaged sampling period that is calculated by averaging the time taken to run a specified number of cycles of the Simulink block diagram. The number of cycles used for averaging is user definable, such that the user can adjust the resolution of the sampling period, i.e., a large number of cycles will provide a finer resolution of the sampling period compared with a small number of cycles. However, this procedure of obtaining a sampling period does not provide the exact sampling period for each Simulink block cycle, thus, this DAC toolbox does not enforce any real-time requirements, i.e., enforcing a specific sampling period for the Simulink block diagram is not permissible.

IV. Simulink Diagram Details

This section describes in detail the sequence of tasks that are performed before running a Simulink block diagram. In particular, we outline the set of tasks that the function, **TotalCompile**, performs. Finally, in this section we will describe the serial data communication between Matlab and BS2.

A. TotalCompile

TotalCompile is composed of a sequence of sequential tasks.

1) Using Global Variables: Global variables are used in order to share data with any sensor or actuator block from the BS2 library.

2) Storing Sensors and Actuators Blocks: After defining global variables the **TotalCompile** function reads the Simulink model file as a text file. In this process the function looks for all the blocks (by name) that are present in the Simulink block diagram that match with the ones stored in the BS2 library. It should be noted that for this task to be performed correctly, the Simulink model file must be saved before starting the Simulink block diagram. Unsaved Simulink model files may not include the most recent changes to the sensor or actuator block parameters. Furthermore, if a user adds or removes sensor or actuator blocks to or from the block diagram, this unsaved Simulink model file may not reflect these recent changes.

When a matching block is found, it is then categorized as a sensor or an actuator. The type of block is then used in conjunction with the specified block parameters to be stored in a data structure depending on its category, e.g., a sensor or an actuator structure. Thus, for multiple sensor blocks in a Simulink block diagram, an array of sensor structures is allocated. Similarly, an array of actuator structures is allocated. The sensor array stores all sensor blocks present in the Simulink block diagram, and an actuator array stores all actuator blocks present in this diagram. If there are no sensor or actuator blocks present in the diagram a “null” object is stored indicating an empty array.

The order in which sensor blocks are stored in the array is the same as the order in which the sensor blocks are listed in the text file of the Simulink model file. The actuator array is organized in a similar manner.

Since in a Simulink block diagram it is possible to have multiple blocks with the same name, two parameters are used to determine each block uniquely. The first parameter is a pin of the sensor/actuator connected to the BS2. However, certain sensors or actuators devices may have multiple channels, in which case, a second parameter is used to store the channel information. The sensor or actuator array information is later used to organize data sent via serial communication.

3) Generating Matlab and PBasic Code: The sensor and actuator arrays are used to generate Matlab and PBasic code. The PBasic code, which is used to program the BS2, is generated first. We note that within the BS2 block library, every sensor or actuator block has an associated PBasic code. Thus, for a given sensor or actuator structure, a corresponding PBasic code can be provided. Finally, the PBasic code is organized as follows: *i)* BS2 waits for Matlab to send the actuator data, *ii)* this actuator data drives the actuators, *iii)* data is gathered from the sensors, and *iv)* sensor data is sent to the Simulink block diagram. Next, a section of the **IOBlock**’s Matlab code is generated to facilitate serial communication between Matlab and BS2. In particular, this Matlab code sends and receives the same amount of data that the BS2 receives and sends, respectively.

*4) Programming the BS2***:** Referring to Figure 3, programming the BS2 involves *i)* the tokenization of the PBasic code and *ii)* the sending of this tokenized code to the BS2 via serial communication. The tokenization process involves sending the PBasic code to a C++ executable program. This program performs tokenization of the PBasic code using a tokenizer library provided by Parallax Inc. and stores the result in a text file. The tokenized code is organized as a set of data packets to be sent to the BS2. See [13] for details on how to tokenize PBasic code. Next, a Java program transmits the tokenized PBasic code packet by packet to the BS2 using serial communication. See [13] for details on how to program the BS2 via serial communication. After attempting to program the BS2, a Boolean is set to true or false depending on the success or failure of programming the BS2, respectively.

5) Starting a Simulink Block Diagram: If the BS2 was successfully programmed, then the Simulink block diagram will start, otherwise the block diagram will stop and produce an error message on the Matlab command window.

B. BS2 and Matlab Serial Communication

Referring to Figure 4, BS2 and Matlab communicate with each other using the serial communication port. Both BS2 and Matlab have built-in functions that provide serial communication capabilities. It is important to note that all sensor data is sent from Matlab as one packet and all actuator data is received by Matlab as one packet through serial communication. Sensor data in the sensor packet can be retrieved by the corresponding sensor blocks in the Simulink block diagram, whereas data from the actuator blocks are packaged into a single packet for transmission to the BS2.

The use of packets for data communication between Matlab and BS2 is efficient compared with the transmission of individual, disjoint sensor or actuator data. In transmitting packets for data communication, the amount of information needed to be sent via serial communication is reduced, i.e., in one sensor or actuator packet, the necessary data for serial communication is one start and stop bit, whereas for individual, disjoint sensor or actuator data, multiple start and stop bits are necessary for serial communication.

The **IOBlock** receives the sensor packet and stores the data in a sensor global variable. The sensor packet received from the BS2 is first converted to the appropriate sensor data, i.e., a numerical value dependent on the sensor type, and is then stored in the sensor global variable. The **IOBlock** transmits data from the actuator global variable to the BS2 for execution. In constructing the actuator global variable, each actuator block converts the numerical value of their actuator data into a set of bytes. Next, the sets of bytes for all actuator blocks are packaged into a packet and saved into the actuator global variable. See Figure 4 for a graphical description of this packet. Lastly, data communication between BS2 and Matlab will continue until the Simulink block diagram is stopped.

C. Data Organization

Both sensor and actuator data transmitted through serial communication is organized in a specific order. As seen in Figure 5, the order in which data is stored for a sensor packet is the same as how the sensor structures are ordered in the sensor array. Furthermore, the actuator packet is ordered in a similar manner.

Each BS2 library block has access to both sensor and actuator arrays and sensor and actuator global variables. Each sensor or actuator used in the Simulink block diagram is uniquely determined based on a pin number that the sensor or actuator uses for connection with the BS2 and the device channel (if device controls multiple channels). Depending on the pin number and device channel, each sensor or actuator block can search for its position in its corresponding sensor or actuator array. See Figure 5 for an example of a sensor block with its corresponding position in the sensor array. Once the position of a sensor block is determined, the appropriate sensor data can be retrieved from the sensor global variable. Also, an actuator block can store actuator data to the actuator global variable in a similar manner.

V. Example – DC Motor Control

For illustrative purposes, in this example, we explore a DC motor control experiment using the BS2 microcontroller and our Matlab DAC toolbox. The DC motor test-bed consists of an armature controlled DC motor, a continuous rotation potentiometer, a tachometer, and a power amplifier. This test-bed, shown in Figure 4, is manufactured by Quanser Consulting Inc. The potentiometer outputs a +/-5VDC signal corresponding to the absolute angular position of the motor. The tachometer outputs a +/-5VDC signal corresponding to the angular velocity of the motor. The BS2 supplies a controlled voltage signal to control the DC motor angular position. The DC motor sends and receives analog signals from the microcontroller using an LTC1296 A2D and a MAX537 D2A, respectively. A MAX764 DC-DC inverter, manufactured by Dallas Semiconductor Inc. [14] and powered by the BOE's +5VDC power supply, is used to obtain a +/-5VDC power supply for the LTC1296 and MAX537.

In this experiment, we used Matlab version 6.5, which has a built-in serial communication library, in addition to Simulink version 5.0. This experiment utilizes the classical proportional-integral-derivative (PID) controller to control the position of the DC motor. Referring to Figure 4, two A2D blocks are used to import sensor data to the PID controller, i.e., the block labeled AtoD_LTC_Pot sends DC motor potentiometer data and the block labeled AtoD_LTC_Tach sends DC motor tachometer data to the PID controller. The output from these sensor blocks is then connected to appropriate calibration gain blocks. Furthermore, a D2A block labeled as DtoA_MAX_Motor is used to transmit the PID controller output to the DC motor.

A. Experimental Results

A classical PID controller is implemented in Simulink to control a DC motor using the BS2 library. Specifically, an analog PID controller is designed and implemented using Simulink's Euler approximation integration algorithm [8] (with a sampling period of 0.13sec). For illustrative purposes, two sets of performance specifications are used to design corresponding PID control gains that are used to obtain experimental response of the motor for 0 degrees and 90 degrees angular position commands for the motor arm.

The PID control gains used in this experiment are computed using the analytical model of the DC motor under the PID feedback control. It can be shown that a third-order transfer function with one real pole, a pair of complex-conjugate poles, and a finite zero captures the closed-loop transfer function of the DC motor with the PID controller (see, Section 5.4 of [15] for a similar transfer function). Next, by specifying the desired damping ratio and natural frequency of complex-conjugate closed-loop poles and the location of the real pole as 0.69, 1.16, and -47.8479, respectively, the PID control gains are computed to be $K_p = 1.28$, $K_I = 1.06$, and $K_D = 0.21$. Following [15], for these control gains it can be shown that the closed-loop response theoretically exhibits a 2 percent settling time of 5sec and a percent overshoot of 25%. Referring to Figure 6, which shows experimental time history of the DC motor arm angular position, these control gains were used from 0 to 58sec. As evidenced from Figure 6, the experimental response exhibits an average experimental 2 percent settling time of 9.89sec and a percent overshoot of 29.36%. Next, by specifying the desired damping ratio and natural frequency of complex-conjugate closed-loop poles and the location of real pole as 0.69, 0.58, and -24, respectively, the PID control gains are computed to be $K_p = 0.32$, $K_I = 0.13$, and $K_D = -0.19$. For these control gains it can be shown that the closed-loop response theoretically exhibits a 2 percent settling time of 10sec and a percent overshoot of 25%. Referring to Figure 6, these control gains were used from 58 to 105sec. As evidenced from Figure 6, the experimental response exhibits an average experimental

2 percent settling time of 11.03sec and a percent overshoot of 31%.

VI. Conclusion

In this paper, we developed an inexpensive data acquisition and control system by exploiting the serial communication capabilities of Matlab and the BS2 microcontroller. Using the advanced features of Simulink, our software environment allows for the generation of PBasic code for a variety of sensors and actuators, programming of the BS2 microcontroller, and data communication between BS2 and Matlab. Furthermore, a DC motor control experiment was conducted to show the salient features of our DAC toolbox. Specifically, a PID controller was implemented in a Simulink block diagram to control the DC motor arm position.

References

[1] Online: http://www.advantech.com/, website of Advantech Co.

[2] Online: http://www.ni.com/, website of National Instruments Corp.

[3] Online: http://www.quanser.com/choice.asp, website of Quanser Consulting Inc.

[4] Online: http://www.mathworks.com/products/simulink/, website of MathWorks Inc., developer and distributor of Simulink.

[5] Online: http://www.ni.com/labview/, website of National Instruments Corp., developer and distributor of LabVIEW.

[6] Online: http://www.parallax.com/detail.asp?product_id=BS2-IC, website of Parallax Inc., developer and distributor of the Basic Stamp 2 (BS2-IC) microcontroller (access link for BS2-IC product information).

[7] Y. F. Li, S. Harari, H. Wong, and V. Kapila, "Matlab-Based Graphical User Interface Development for Basic Stamp 2 Microcontroller Projects," *Proceedings of the American Control Conference*, Boston, MA, pp. 3233–3238, 2004.

[8] Online: http://www.mathworks.com/access/helpdesk/help/pdf_doc/simulink/sl_using.pdf, website of MathWorks Inc., developer and distributor of Simulink (access link for documentation on Simulink callback functions).

[9] Online: http://www.parallax.com/dl/docs/cols/nv/vol1/col/nv15.pdf, website of Parallax Inc., developer and distributor of the Basic Stamp 2 microcontroller (access link for documentation on the RCtime instruction for the Basic Stamp 2 microcontroller).

[10] Online: http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1155,C1001,C1158,P1484,D3526, website of Linear Technology Corp., developer and distributor of the LTC1296 A2D (access link for product information).

[11] Online: http://www.parallax.com/dl/docs/prod/motors/stdservo.pdf, website of Parallax Inc., of the Basic Stamp 2 microcontroller (access link for servo motor product information).

[12] Online: http://pdfserv.maxim-ic.com/en/ds/MAX536-MAX537.pdf, website of Dallas Semiconductor Inc., developer and distributor of the MAX537 D2A (access link for product information).

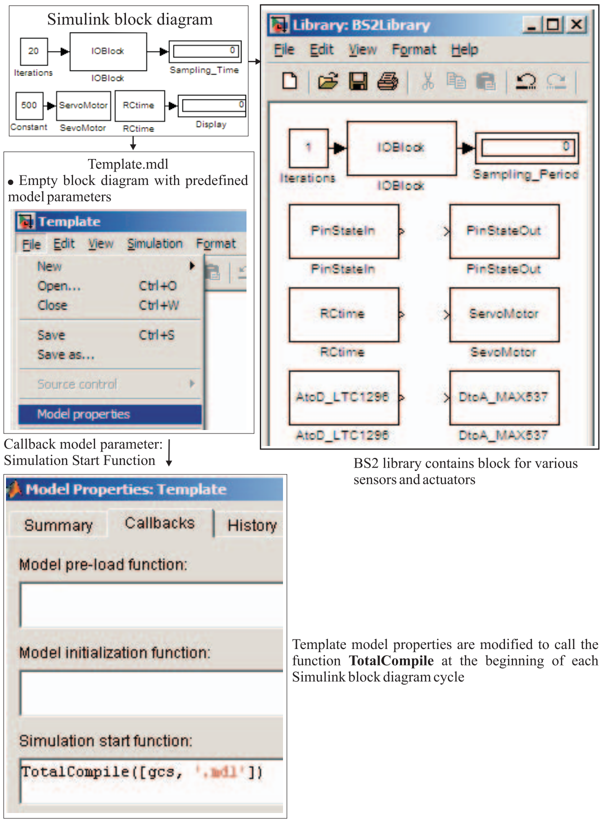
[13] Online: http://www.parallax.com/html_pages/downloads/tokenizer/tokenizer.asp, website of Parallax Inc., developer and distributor of the PBasic tokenizer library (access link for the Parallax tokenizer library and the tokenizer documentation).

[14] Online: http://pdfserv.maxim-ic.com/en/ds/MAX764-MAX766.pdf, website of Dallas Semiconductor Inc., developer and distributor of the MAX764 DC-DC inverter (access link for product information).

[15] R. C. Dorf and R. H. Bishop, *Modern Control Systems*. Addison Wesley, Menlo Park, CA, 2005.

TABLE I:
Sensor and Actuator Block Description

Name	Description
PinStateIn	Sensor block giving the state of a BS2 pin, which is either high (5Volts) = 1 or low (0Volts) = 0
RCtime	Sensor block that measures the time it takes for a pin to change its state from high (5Volts) to low (0Volts), or vice versa
AtoD_LTC1296	Sensor block that provides voltage on a specified channel from an LTC1296 analog to digital converter (A2D)
PinStateOut	Actuator block which changes the state of a BS2 pin to high (5Volts) = 1 or low (0Volts) = 0
ServoMotor	Actuator block that turns a servo motor to a desired position
DtoA_MAX537	Actuator block that outputs a specified voltage to a MAX537 digital to analog converter (D2A)



Template.mdl
• Empty block diagram with predefined model parameters

Callback model parameter:
Simulation Start Function

BS2 library contains block for various sensors and actuators

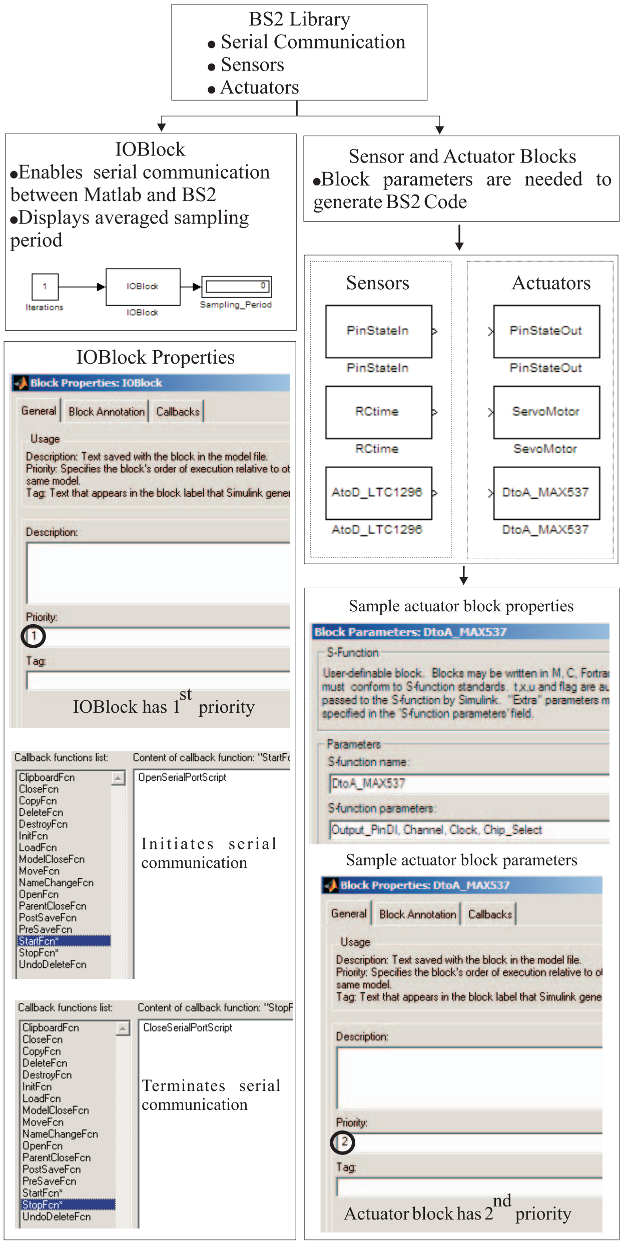
Model pre-load function:

Model initialization function:

Simulation start function:
```
TotalCompile([gcs, '.mdl'])
```

Template model properties are modified to call the function **TotalCompile** at the beginning of each Simulink block diagram cycle

Figure 1: Matlab data acquisition and control toolbox software interface



BS2 Library
- Serial Communication
- Sensors
- Actuators

IOBlock
- Enables serial communication between Matlab and BS2
- Displays averaged sampling period

Sensor and Actuator Blocks
- Block parameters are needed to generate BS2 Code

IOBlock Properties

Block Properties: IOBlock

Usage

Description: Text saved with the block in the model file.
Priority: Specifies the block's order of execution relative to ot
same model.
Tag: Text that appears in the block label that Simulink gene

Description:

Priority: 1

Tag:

IOBlock has 1st priority

Callback functions list: Content of callback function: "StartF

OpenSerialPortScript

Initiates serial communication

Callback functions list: Content of callback function: "StopF

CloseSerialPortScript

Terminates serial communication

Sample actuator block properties

Block Parameters: DtoA_MAX537

S-Function

User-definable block. Blocks may be written in M, C, Fortra
must conform to S-function standards. t,x,u and flag are au
passed to the S-function by Simulink. "Extra" parameters m
specified in the 'S-function parameters' field.

Parameters

S-function name: DtoA_MAX537

S-function parameters: Output_PinDI, Channel, Clock, Chip_Select

Sample actuator block parameters

Block Properties: DtoA_MAX537

Usage

Description: Text saved with the block in the model file.
Priority: Specifies the block's order of execution relative to o
same model.
Tag: Text that appears in the block label that Simulink gene

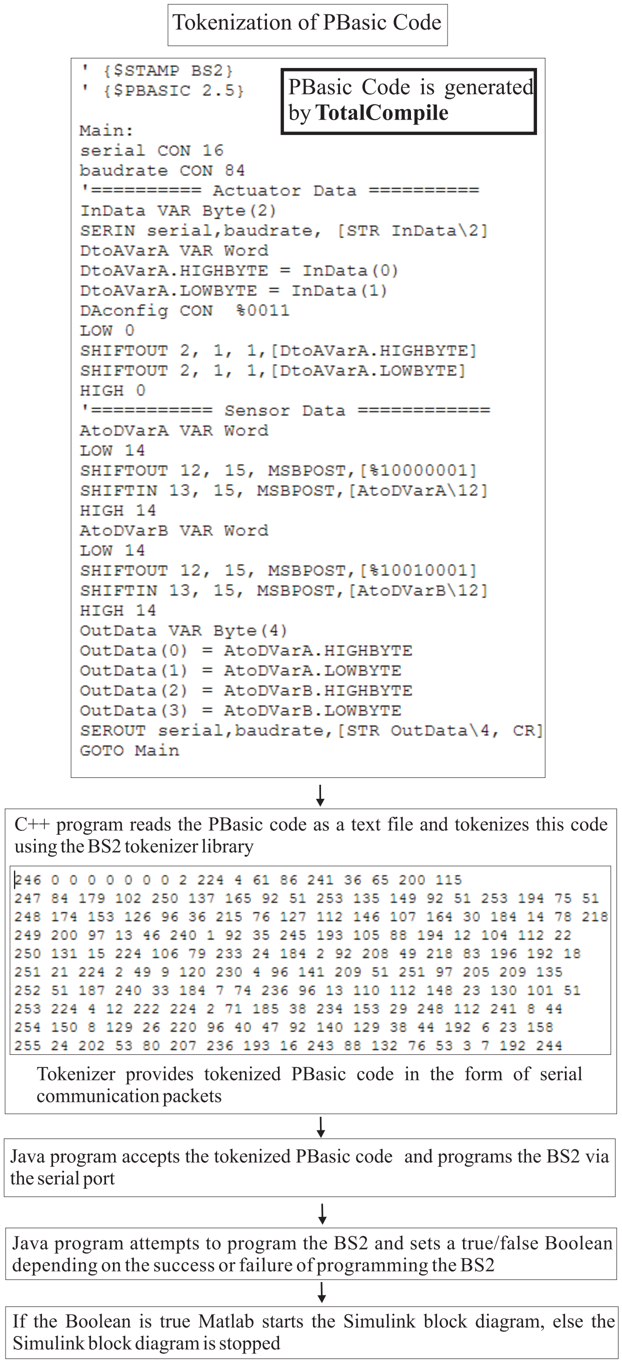
Description:

Priority: 2

Tag:

Actuator block has 2nd priority

Figure 2: Graphical description of the BS2 library



Tokenization of PBasic Code

PBasic Code is generated by **TotalCompile**

```
' {$STAMP BS2}
' {$PBASIC 2.5}

Main:
serial CON 16
baudrate CON 84
'========== Actuator Data ==========
InData VAR Byte(2)
SERIN serial,baudrate, [STR InData\2]
DtoAVarA VAR Word
DtoAVarA.HIGHBYTE = InData(0)
DtoAVarA.LOWBYTE = InData(1)
DAconfig CON  %0011
LOW 0
SHIFTOUT 2, 1, 1,[DtoAVarA.HIGHBYTE]
SHIFTOUT 2, 1, 1,[DtoAVarA.LOWBYTE]
HIGH 0
'=========== Sensor Data ============
AtoDVarA VAR Word
LOW 14
SHIFTOUT 12, 15, MSBPOST,[%10000001]
SHIFTIN 13, 15, MSBPOST,[AtoDVarA\12]
HIGH 14
AtoDVarB VAR Word
LOW 14
SHIFTOUT 12, 15, MSBPOST,[%10010001]
SHIFTIN 13, 15, MSBPOST,[AtoDVarB\12]
HIGH 14
OutData VAR Byte(4)
OutData(0) = AtoDVarA.HIGHBYTE
OutData(1) = AtoDVarA.LOWBYTE
OutData(2) = AtoDVarB.HIGHBYTE
OutData(3) = AtoDVarB.LOWBYTE
SEROUT serial,baudrate,[STR OutData\4, CR]
GOTO Main
```

C++ program reads the PBasic code as a text file and tokenizes this code using the BS2 tokenizer library

```
246 0 0 0 0 0 0 0 2 224 4 61 86 241 36 65 200 115
247 84 179 102 250 137 165 92 51 253 135 149 92 51 253 194 75 51
248 174 153 126 96 36 215 76 127 112 146 107 164 30 184 14 78 218
249 200 97 13 46 240 1 92 35 245 193 105 88 194 12 104 112 22
250 131 15 224 106 79 233 24 184 2 92 208 49 218 83 196 192 18
251 21 224 2 49 9 120 230 4 96 141 209 51 251 97 205 209 135
252 51 187 240 33 184 7 74 236 96 13 110 112 148 23 130 101 51
253 224 4 12 222 224 2 71 185 38 234 153 29 248 112 241 8 44
254 150 8 129 26 220 96 40 47 92 140 129 38 44 192 6 23 158
255 24 202 53 80 207 236 193 16 243 88 132 76 53 3 7 192 244
```

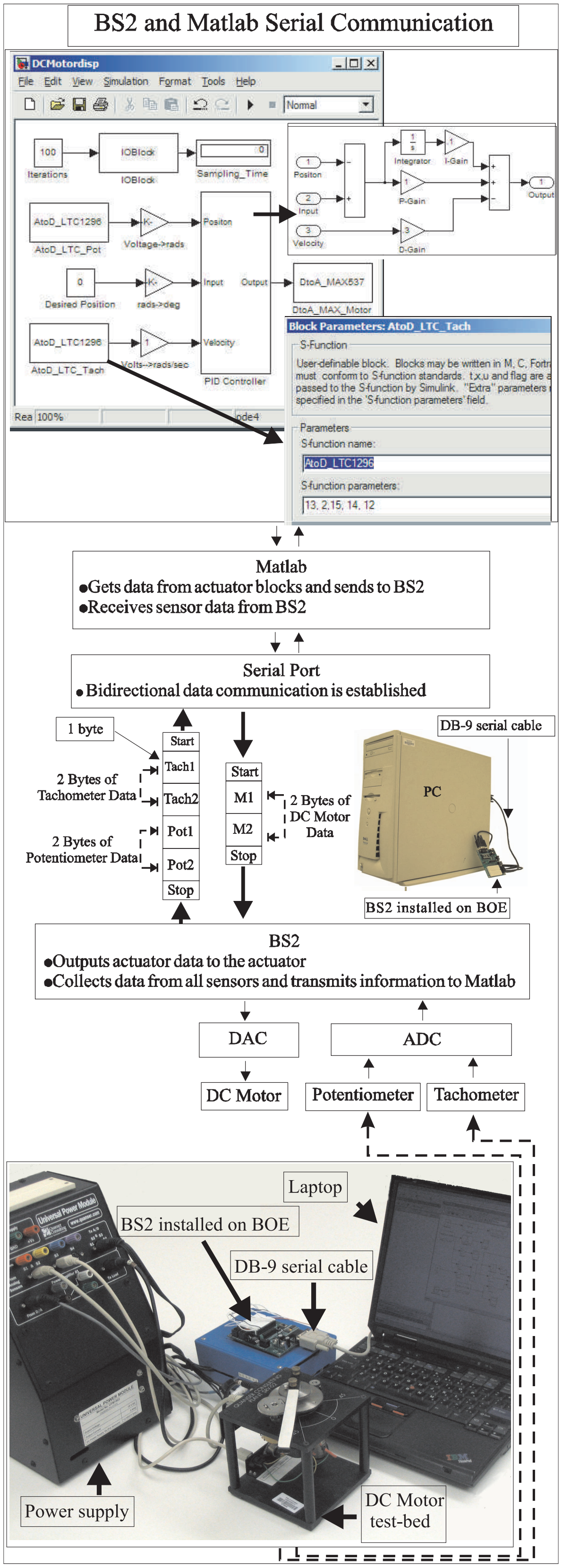
Tokenizer provides tokenized PBasic code in the form of serial communication packets

Java program accepts the tokenized PBasic code and programs the BS2 via the serial port

Java program attempts to program the BS2 and sets a true/false Boolean depending on the success or failure of programming the BS2

If the Boolean is true Matlab starts the Simulink block diagram, else the Simulink block diagram is stopped

Figure 3: Flow diagram for programming the BS2 microcontroller



User-definable block. Blocks may be written in M, C, Fortr
must confom to S-function standards. t,x,u and flag are a
passed to the S-function by Simulink. "Extra" parameters
specified in the 'S-function parameters' field.

S-function name: AtoD_LTC1296

S-function parameters: 13, 2,15, 14, 12

Matlab
●Gets data from actuator blocks and sends to BS2
●Receives sensor data from BS2
Serial Port
●Bidirectional data communication is established
1 byte
Start
Tach1
Tach2
Pot1
Pot2
Stop
2 Bytes of Tachometer Data
2 Bytes of Potentiometer Data
Start
M1
M2
Stop
2 Bytes of DC Motor Data
DB-9 serial cable
PC
BS2 installed on BOE
BS2
●Outputs actuator data to the actuator
●Collects data from all sensors and transmits information to Matlab
DAC
ADC
DC Motor
Potentiometer
Tachometer
Laptop
BS2 installed on BOE
DB-9 serial cable
Power supply
DC Motor test-bed

Figure 4: Graphical description of serial communication between BS2 and Matlab

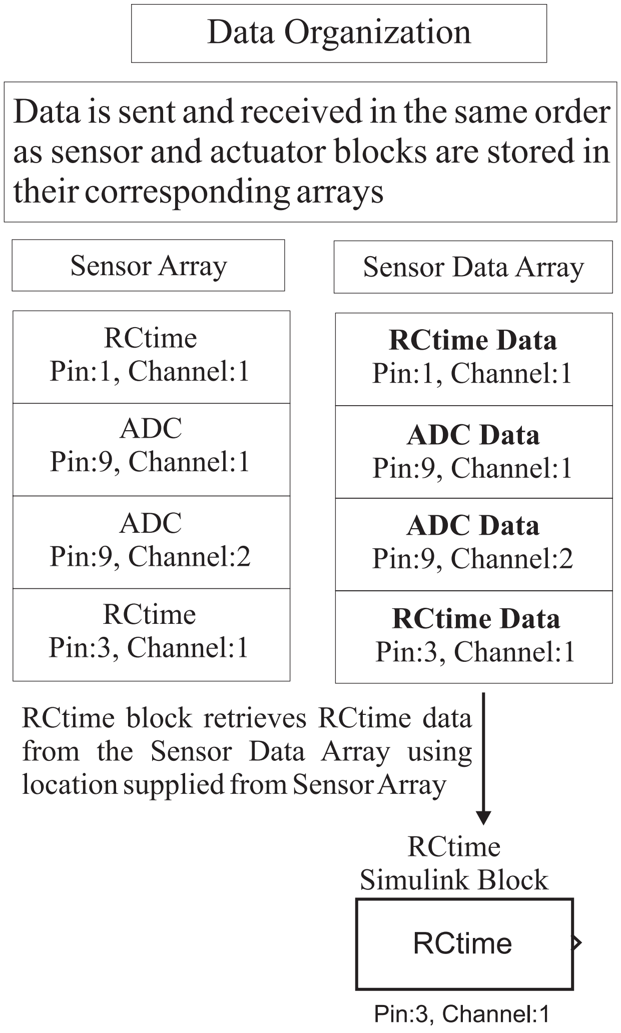

Data Organization
Data is sent and received in the same order as sensor and actuator blocks are stored in their corresponding arrays
Sensor Array
Sensor Data Array
RCtime Pin:1, Channel:1
RCtime Data Pin:1, Channel:1
ADC Pin:9, Channel:1
ADC Data Pin:9, Channel:1
ADC Pin:9, Channel:2
ADC Data Pin:9, Channel:2
RCtime Pin:3, Channel:1
RCtime Data Pin:3, Channel:1
RCtime block retrieves RCtime data from the Sensor Data Array using location supplied from Sensor Array
RCtime Simulink Block
RCtime
Pin:3, Channel:1

Figure 5: Example for sensor array data organization

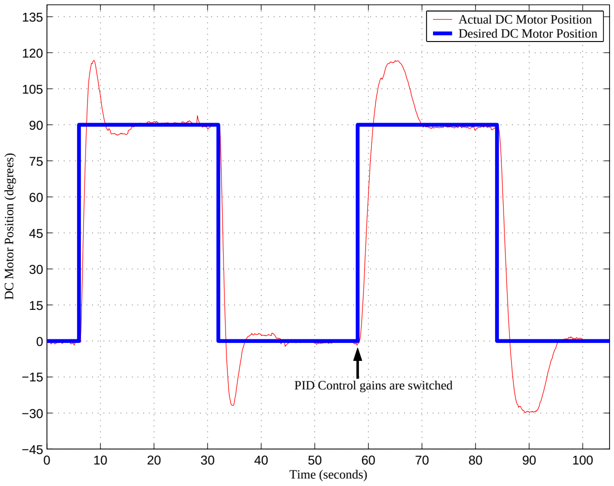


Figure 6: DC motor position tracking response