NOTES ON BUILDING REAL-TIME APPLICATIONS WITH MATLAB

Kostas S. Tsakalis
August 25, 2004

Abstract
The real-time workshop (RTW) of SIMULINK and the xPC toolbox offer exciting capabilities for the design and construction of embedded projects. These can be used for R&D, rapid prototyping, as well as education purposes. In this note, we discuss the programming details for the development of educational projects in control systems.

1. Hardware and Software Platforms

1.1 CPU
The main hardware component is Advantech's PC-104 PCM-3350F board. It has a Pentium-class, 300MHz processor, an Intel 82559 chipset for TCP/IP Ethernet communications and two RS-232 serial ports. The board is equipped with a 128MB flash memory and 128MB RAM. Apart from its sheer computing power, this board has two highly desirable characteristics: one is that it can be configured to boot from the flash memory (truly compact and standalone) and the other is that it is compatible with SIMULINK RTW.

1.2 ADC/DAC
The second hardware component is a Diamond-MM, PC-104 data acquisition module. It has 16 single-ended A/D analog input channels and 2 D/A analog output channels, both with 12-bit resolution. It can operate at 10kHz (100kHz with direct DMA transfer), which far exceeds the requirements of this project (2Hz sampling). This board has low power requirements (5V, 165mA), implying that the entire embedded system can be powered by a very modest (and small) power supply. It is also fairly inexpensive and, it goes without saying, compatible with SIMULINK RTW. Its two drawbacks are in the analog outputs. It has the usual two D/A channels and there is no bipolar mode. The former could present a problem if our project calls for more analog outputs.

1.3 Other hardware
The rest of the hardware components are fairly trivial (only if you already have them):
- An RS-232 crossover cable and an Ethernet crossover cable. The second is only needed for the Matlab communication with the PC-104, especially in a DOS-Loader mode. This mode is unquestionably superior during debugging where the Ethernet connection is a must; the RS-232 is too slow even at its highest speed of 115200 Bps.
- One USB serial/parallel port replicator. This component is necessary with the modern laptops that have no serial or parallel ports. For this, Keyspan's mini port replicator worked flawlessly and its serial port monitor proved extremely valuable during debugging. The Interrupt (compatible) driver setting was used.
- A PC-104 terminal board (to connect the analog inputs and outputs).

1.4 Software
The software platform is MATLAB's R13 with the xPC Target and xPC Embedded Target Toolboxes. Microsoft's Visual Studio 6.0 or Watcom 11 can be used as compilers. Watcom 10.6 and occasionally Microsoft's Visual C run into code-size limitations with large models. The former has internal limitations while the latter recommends the use of a "large model" option in the compiler (this requires expert editing of MATLAB's batch files). Notice that large models fail to compile, even with Watcom 11. Release 13 is apparently important since the use of RS-232 communications on the laptop PC side requires a patch. MATLAB claims that there is no known fix for the previous versions.

2. Creation of Real-Time executable code in SIMULINK/RTW

Once the model is coded in Simulink, the creation of a real-time executable code promises to be a fairly automatic and painless process. However, this is accurate only in the later stages of development where most of the details have been ironed out. This is also exactly the point where each application presents its own individual difficulties that are rarely covered in the standard manual examples. We focus on a D/A conversion to provide analog outputs, an A/D conversion to check the results and assess quantization issues, and RS-232 serial communication with the host PC.

2.1 ADC/DAC

The analog I/O blocks present no difficulty, at least after reading the manual and correctly configuring the board jumpers. The Diamond drivers contained in the xPC Toolbox worked flawlessly, although they do not exploit all the capabilities of the board.

2.2 Serial Communication

The RS-232 serial communication functions present problems both in the embedded code (xPC) and the host PC code (MATLAB-commands).

On the xPC side the main issue is the apparent incompatibility of the recommended v.2 RS-232 with "s-functions" in the Simulink model. (S-functions and v.2 RS-232 create terminal errors at compilation or link.) The v.1 RS-232 functions from the xPC toolbox appear to offer a solution. Other than that, serial communication in xPC is reasonably smooth.

On the host PC side, the RS-232 drivers are implemented in Java and are available only for MATLAB V.6 (R12) or later. For the earlier versions, these drivers were flawed and would freeze the PC. At this point (June 2003) new functions are available for R13, fixing this problem but should be installed by the user. (For later versions, e.g., R14, these problems should be fixed without patches.) A second issue is the inability of the drivers to handle too many bytes at a time. This is not related to any lack of time to perform the transfer. Instead, it is likely that this is a Windows limitation. Of course, Windows is not a real-time operating system and the various background services tend to make it very unpredictable for real-time tasks. One remedy is to break down a long message into shorter ones and use several transmission/reception blocks with subsequent decoding of the received messages. Also, during a run, multiple Windows tasks should be avoided and mechanisms should be implemented to deal with missing data.

2.3 S-functions

S-functions in Simulink can speed-up significantly the compilation, execution and target-host communication of models. This process is fairly quick and painless (it is an RTW option) and reduces considerably the time to create the final embedded real-time code. It also provides a 10x speed-up of non-real-time execution in our application. When the model contains custom s-functions, the build procedure becomes a little more complicated. Using the "s-function builder" block, s-functions can be defined in a somewhat straightforward manner. This block offers the ability to produce automatically .tlc files that are necessary for the creation of real-time standalone executables. The intricacy in coding these functions comes from passing the function arguments as pointers. This is a SIMULINK convention and a careful reading of the manual or on-line help is a must. Once this is resolved, a moderately experienced C programmer can easily code and compile custom code. No further adjustments are necessary to produce standalone executables with s-function builder blocks that appear on the "top layer" of SIMULINK (make sure the TLC generation option is checked). On the other hand, when these s-function builder blocks are nested inside RTW S-functions (s-function blocks created by an RTW auto-coding process) it is necessary to adjust the SIMULINK build parameters to reflect the use of additional objects during the link process. Failure to do so results in a message **`"undefined symbol filename_Outputs_wrapper."`**
This function is defined by the s-function builder block inside the file "filename_wrapper" and this is exactly the file that must be added as a link object. The easiest way to do this is to adjust the parameters of the SIMULINK block that contains the user-defined function. The command for this is
`set_param('SfunctionObject','SFunctionModules','filename_wrapper')`

where “SfunctionObject” is the complete name of the s-function containing the custom code and “filename” is the name of the custom s-function created by the s-function build block (multiple file names can be entered, separated by space).

2.4 Summary of the Procedure to Build Real-Time Executables

With the above comments, the creation of a real-time executable becomes routine and painless, as promised:

1. **Run xpcsetup to define the xPC communication and auto-coding parameters.** Use small or medium “max. code size” (2-4MB) since the 16MB option is unsupported for embedded standalone codes. The standalone option (requires xPC embedded Target) or DOS-Loader are suitable for PC-104 boards booting from the flash disk.
2. **Select the communication parameters, preferably TCP/IP.** For RS-232, only COM1 and COM2 can be used (fortunately, the Keyspan driver can be reconfigured to use COM2). Of course, the RS-232 choice prohibits the use of the serial port for transmitting any other messages and it was hard and expensive enough to get one COM port in our laptop. For the TCP/IP setup, make sure that the laptop host has a fixed address. Then increment it by one for the PC-104. E.g., use 169.192.1.0 in the host, 169.192.1.1 in the target and 255.255.0.0, for TCP/IP subnet mask in both. Then, use 223.255.0.0 for the Gateway in the PC and 255.255.255.255 in the target, signifying direct connection. Also make sure that the correct chipset is selected (Advantech uses the I52559 and the bus is PCI).
3. **For the DOS-Loader mode, create a boot disk.** This requires a DOS booting disk, which is a straightforward task in Win98 or earlier. It cannot be made in Win NT, XP etc. The program performing this function is an m-file but expects the boot disk in A:. This is a minor inconvenience that should have been anticipated (laptops rarely come with a disk drive any more). Simply replace every occurrence of “A:” with “E:” (or the appropriate drive letter for the flash reader in the laptop). Use the boot disk (i.e., flash) and startup the PC-104. If a monitor is unavailable, issue successive “xpctargetping” commands at the Matlab prompt to make sure that the PC-104 has finished booting.
4. **Open the Simulink model file containing the embedded system.** Make sure that the simulation parameters are set for Discrete, and in the RTW pane, the xPCTarget makefile is selected. Defaults for the rest work fine. Also make sure that any parameters are initialized in the Matlab command window, as necessary.
5. **Right-click on the model core, representing the bulk of the computational load, and select “RTW>Create S-Function.”** The driver will prompt for parameters to be made external. No selection is necessary, unless the code is to be reused with minor adjustments. If need be, the S-function creation can always be repeated from scratch. After successfully completing the build process, the S-function will appear in a new window, having exactly the same size and I/O structure as the original model. Using cut and paste, replace the original Simulink block with the S-function block and save the new model (make sure you save a copy of the Simulink model in a separate file).
6. **If there are custom s-functions nested inside the block converted to an RTW S-function, return to the command window and set its link parameters.** (This step is not necessary if the RTW S-function conversion is only to be used for faster simulation within SIMULINK.) In our case, the model file name is “sfurnace_v2.mdl” and the RTW S-function has the name “Sfurnace_v2.1” which is by default the same as the original block but can be edited as desired. This block is masked to have the same size and appearance as the original block. Right click on this block to see under the mask and get the name of the masked block. By default, this is the same as the original block with the extension “_sfcn”. Again, this name can be edited and in our case is “Sfurnacev21_sfcn”. This function contains the user-defined s-function “s_sew”. Then, at the command window issue the following command to include the “s_sew_wrapper” object code in the link process:

   ```
   set_param('sfurnace_v2/SFurnace_v2.1/SFurnacev21_sfcn','SFunctionModules','s_sew_wrapper')
   ```

7. **Build the embedded real-time code.** For a standalone mode, it is sufficient to enter CTRL-B at the Simulink model window. This will create a <filename>_xpc_emb folder with the three files to be transferred to the flash disk. This disk must be a DOS booting disk but should not contain the DOS-Loader files.

8. **For the DOS-Loader option, an alternative to CTRL-B build is to run the "xpcrctool" function at the command window.** This will open the monitoring tool in a new window. Clicking on the "build model" button will initiate the build process, ending with the download of the executable code to the PC-104. Using the "xpcrctool" the program can then be executed, stopped and the results can be graphed from the host PC.

3. Example of Communication with the Host PC

The RS-232 serial communication with the host PC finds application in the problem where a PC-resident program controls a process that is operated or simulated externally. Input/output measurements are communicated back and forth through the serial port (e.g., with the use of smart sensors and actuators).

The block diagram of such a controller is shown in Fig. 4. This model generates the commands, views and stores data and communicates with the target. The communication with the target is performed by the "Furnace Communication" block, whose components are shown in Fig. 5. The main communication is performed by the MATLAB function block while the rest of the blocks perform multiplexing and de-multiplexing of the various signals. The communication function is written as an ".m file," accepting two inputs, providing one output, and using the MATLAB serial commands to access the serial port.

There are two important issues that this function must resolve:

- One is to make sure that the output vector contains the most recent message; this is achieved in an initialization step reading all messages in the RS-232 buffer (this step is performed only the first time the function is executed).
- The second is to make sure that the transmission port is empty before sending the control signals to the target.

The block diagram of the control algorithm subsystem is shown in Fig. 6. It includes 5 decoupled PID's and a simple mechanism to handle missing data or communication errors. More specifically, if the communication function does not receive data correctly, it outputs a zero measurement vector. When this is detected at the PID input, the last measurement is transmitted to the PID's in order to preserve the continuity of the control input.

Finally, the read/write message formats should be consistently defined in both the target and host. The source codes for the communication function and the target RS-232 message formats are given in the Appendix. At this point, it is convenient to normalize all transmitted data to contain a fixed number of bytes. Using the C '8f' format, each number is transmitted as a floating point with 6 decimals. Thus, using a scaling parameter so that all numbers are less than 10, guarantees that all numbers are transmitted as 8-byte messages plus 1-byte comma separators. The 25 variables will then be transmitted in 225 bytes including 1 byte for the "#" terminator. Fixing the size of the message is not crucial for its transmission or reception. But it helps to identify if a complete message has been received by simply checking the "BytesAvailable" variable.

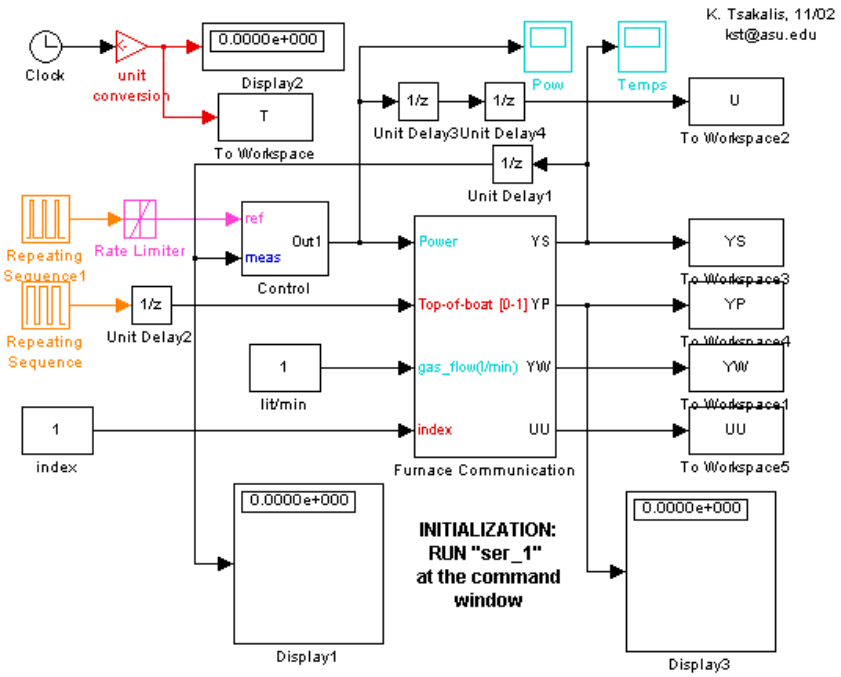

Figure 1: Block diagram of the Controller top layer.

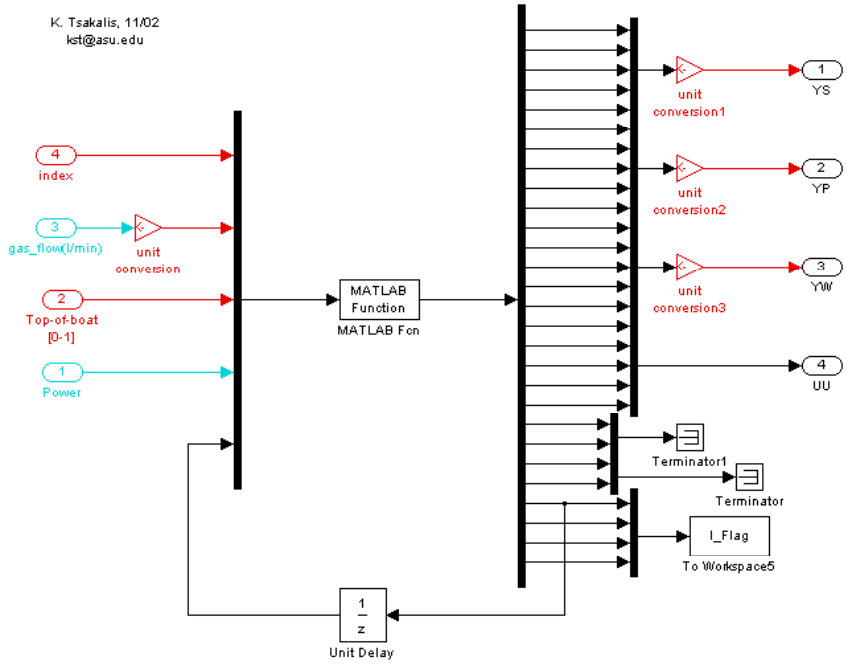

Figure 2: Block diagram of the Controller serial communication subsystem.

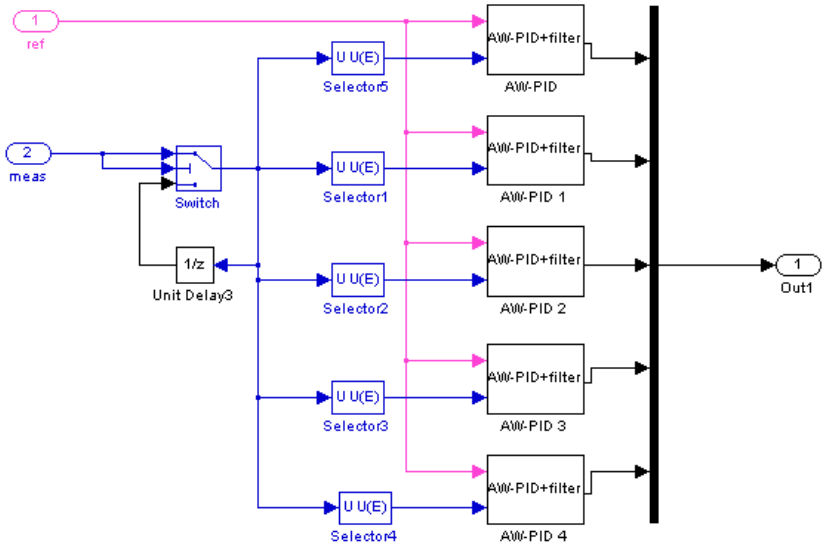

Figure 3: Block diagram of the controller subsystem, including a simple missing data handling mechanism.

APPENDIX

A1. Controller RS-232 communication function source code.

```
function xp=fur_com1(U,s_2);
% Furnace communication function
% Inputs: Control vector, Serial object
% Outputs: Output vector (Temperatures, applied power, flags)
% KST 7/7/03

t0=clock;
g5='%8f,%8f,%8f,%8f,%8f,';g3='%8f,%8f,%8f';      % R/W format
g5e='%8f,%8f,%8f,%8f,%8f';
xp0=zeros(25,1); xp1=xp0;                        % Init
x0=sprintf([g5,g5,g5,g5,g5e,'#'],xp0);
Init_flag = U(length(U));                        % Flags
if Init_flag == 0                                % Init RS232
   while s_2.BytesAvailable < 225 & etime(clock,t0) < 5; end
   while ~isempty(x0)                            % Empty buffer
      x1=x0;
      x0=fscanf(s_2);
   end
   xp0=sscanf(x1,'%f,');                         % last value
   Init_flag=1;
else
   while s_2.BytesAvailable < 225 & etime(clock,t0) < 5; end  % Read message
   x0=fscanf(s_2);
   xp0=sscanf(x0,'%f,');
   if s_2.BytesAvailable >0; Init_flag=0;end    % Enforce empty buffer
end
if length(xp0) ~= 25;   xp0 = zeros(25,1);     Init_flag=0;  end

Up=sprintf([g5,g3,'#'],[U(1:length(U)-1)]);
ez=0;t1=clock;
while ez == 0                                    % Check for busy TX port
   b=s_2.TransferStatus;  dt = etime(clock,t1);
   ez = (dt >= 0.2) + (length(b) == 4);        % Idle or Read status
end
if dt < 0.2; fwrite(s_2,Up,'async'); end        % Send control message
dt1=etime(clock,t1);
dt0=etime(clock,t0);
xp = [xp0(2:25)*Init_flag;Init_flag;0.2-dt;dt1;dt0];
```

A2. Controller RS-232 initialization.

```
%sample code for serial communication
warning off MATLAB:serial:fscanf:unsuccessfulRead
s_2 = serial('COM2','BaudRate',115200,'FlowControl','software','Timeout',.05);
s_2.InputBufferSize=1024;  s_2.OutputBufferSize=1024;  s_2.Terminator=35;
fopen(s_2);
% initialize controller parameters
Tsa=0.5;  decim=1;  PID=[.02 100 100/4];
```