RTW SUPPORT FOR LOW COST C31 BOARD

Christian Vialatte, Jiri Kadlec

UTIA AV CR, Prague, CZ

Introduction

The IDA board is an I/O card built around a C31 DSP processor and connected to a host PC via a RS232 link. It is possible to target SIMULINK models compiled using RTW to this board and execute them in real-time. This brings the card real-time and I/O capabilities to SIMULINK models in which the simulated input and output signals can be replaced by real ones. SIMULINK scopes and outputs can be saved into MATLAB *.mat* files for 'post-mortem' analysis.

This paper presents software support for the Real Time Workshop (Matlab 5.3), targeting the IDA board. The board is using the floating-point Texas Instruments DSP processor TMS320C31, operating on 60MHz. This low cost board includes 2 A/D and 2 D/A 8bit I/O lines with 8 bit resolution, and up to 200kHz sampling rate and 8 digital I/O lines. The board communicates with PC via standard RS232 line.

We will present the performance achieved, basic I/O block library and the PC support for the post-mortem Matlab analysis.

Fig. 1 presents the block diagram of IDA board. The board is implemented as a stay-alone 100x160 mm format Module connected to any PC or portable via serial line.

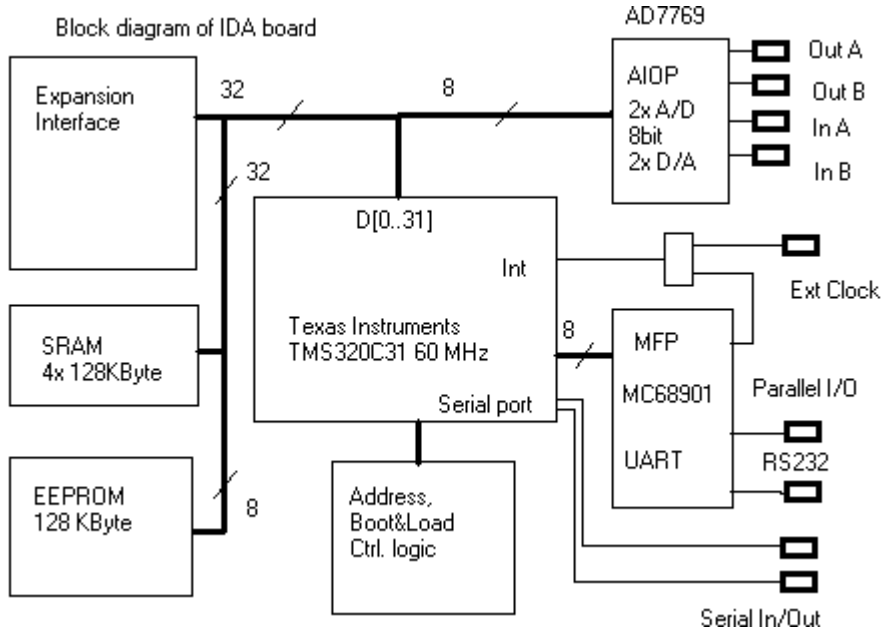

Fig 1. Low Cost C31 IDA board

Installation of RTW support for the IDA board

The installation of the RTW suport fir IDA board is relatively strighforward.

- Execute the self extractable *ida_rtw.exe* file from the floppy disk.
  It will create the *c:\ida_rtw* directory and copy all the files onto your hard disk

The files mentioned below can be found in the *install* sub-directory.

- Copy *nmake.exe* into a directory on your path
  (as defines in the PATH variable in your *autoexec.bat*)

- Copy *mttty.exe* into a directory on your path. See Fig. 3.

- Copy *make_rtw_dsp.m* and *rtw_c_dsp.m* files to *matlab\toolbox\rtw*

- Edit the *grt_c31.tmf* and *grt_c31_notimer.tmf* files if you need to change their default values. Near the top of these files are declared some macros defining the paths to IDA tools,
  C31 C compiler and RTW IDA files. the defaults are:
  - IDA_DIR = c:\ida
  - CV_RTW_DIR = c:\ida_rtw
  - TMSC31_DIR = c:\tms320.c3x

- Copy both *grt_c31.tmf* and *grt_c31_notimer.tmf* to the *matlab\rtw\c\grt* directory

Fig. 2. Presents a simple model, which adds 2 signals and plots the result on a scope. The s-function block s_out serves as the output library block for the IDA board. The s-function code is a see-through function if used by Simulink. Identical code, compiled by the C31 compiler will add output to the A/D port A of IDA board. The Sampling is set to 10000 samples per second (100 microsecond per sample) and the run is executed for 20 secoconds.

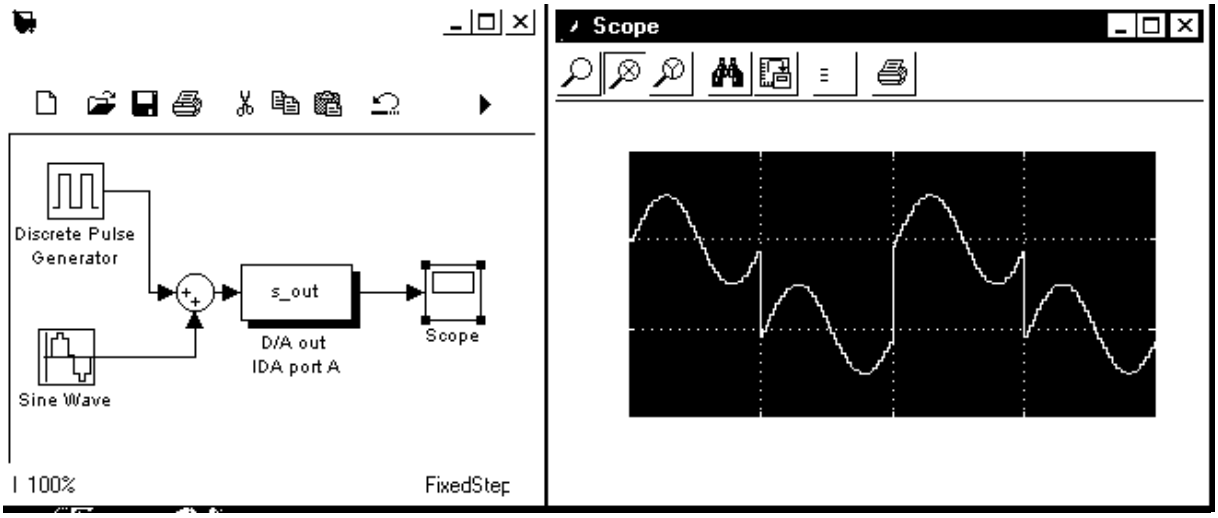

Fig. 2 Simulink block adds 2 signals and plots the result on a scope. The s-function code is a see-through function if used by Simulink. Identical code, compiled by the C31 compiler will add output to the A/D port A of IDA board. Simulink plot presents last 200 samples of total 200k samples. These data are stored to Matlab.

Compiling a SIMULINK model

- Open MATLAB and a SIMULINK model (see Fig. 2).
  The length of the model name must not exceed 7 characters. This is due to the file name length limit of the C31 compiler.

- Use the `cd` command from the MATLAB prompt to go to the directory where you want to compile the model.

- Open the *'Tools' 'RTW Options'* dialogue box and make sure that:

- *'Solver'* type: *Fixed-step*
- *'Solver'* mode: *Single Tasking*
- *'Real Time Workshop' 'system Target File'*: *grt.tlc*
- *'Real Time Workshop' 'Template makefile'*: *grt_c31.tmf* or *grt_c31_notimer.tmf*
- *'Real Time Workshop' 'Make command'*: *make_rtw_dsp*
- *'Real Time Workshop' 'Options' 'External mode'*: unchecked
- *'Real Time Workshop' 'Options' 'Local Block Output'*: unchecked

PC and DSP applications can be generated in the same build directory without any conflicts. The parameters for PC compilation using the PC C compiler registered with MATLAB (typically Microsoft Visual C or Watcom C):

- *'Real Time Workshop' 'Template makefile'*: grt_default_tmf
- *'Real Time Workshop' 'Make command'*: make_rtw

The *grt_c31.tmf* file is used to generate an application using one timer to trigger each sampling step, while *grt_c31_notimer.tmf* builds an application running ‘as fast as possible’. This is useful to find out the minimum time needed to execute each step.

- Type *ctrl-B* in the SIMULINK window to generate the code (the C31 Texas Instrument C3x compiler and IDA tools must be installed)

Data saving must always been done in the ‘Matrix’ format (not structure), this means that multi-channels scopes are not supported. A way around is to use a multiplexer to pack several signals on one line or just use several scopes.

Running the generated application

- Run the *mttty.exe* application (see Fig. 3)
  or
  press F3 to disconnect *mttty* (if you have already executed an IDA application)
- Reset the IDA board
- Wait around 3 seconds after the green led stopped flashing
- F4 to connect
- F5 to load an application
- Browse to choose the application to load
- Reset the IDA board and click 'open' while the green led is flashing

When the program asks for simulation time, type:
`-1 <ENTER>`
to use the model default execution time.

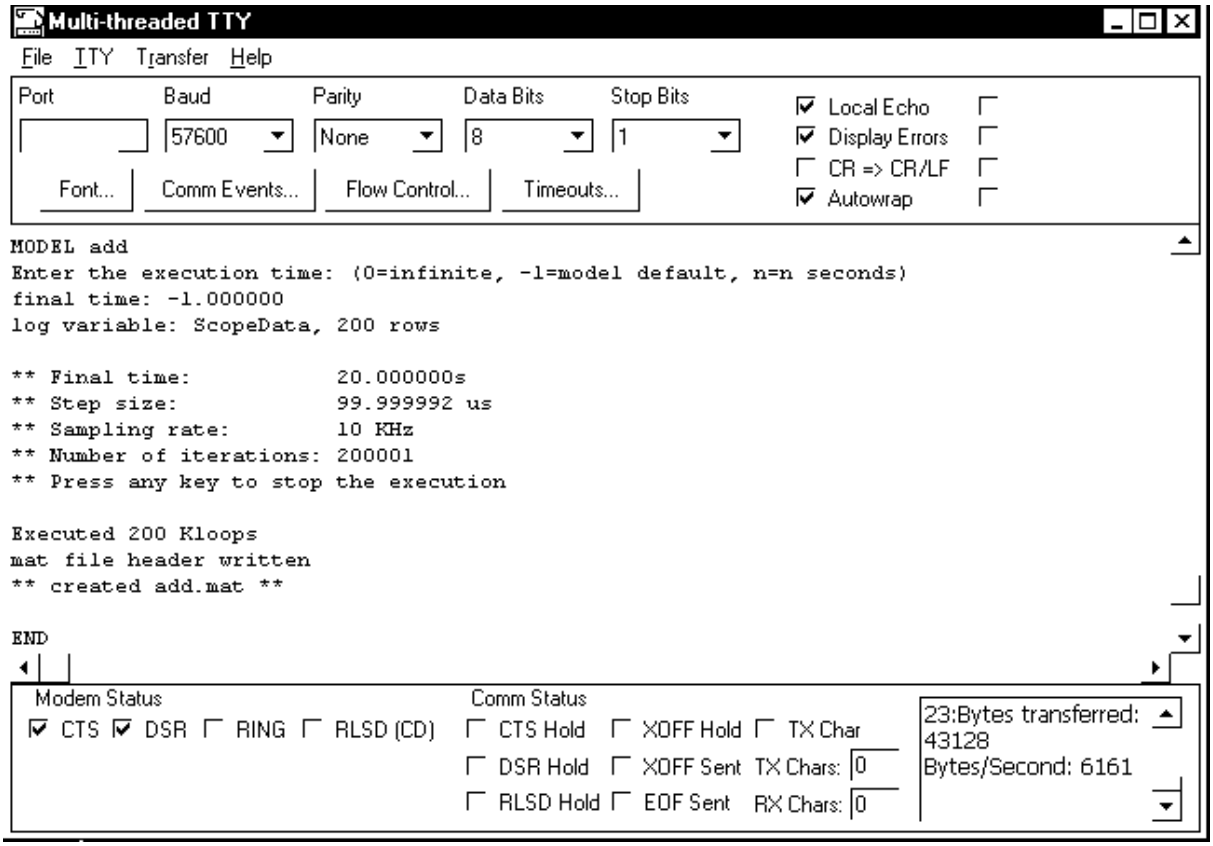

Fig. 3 PC interface for download of IDA and communication. The screen presents the result of execution of the model from Fig. 2 on the IDA board. The output is sampled 10KHz and can be observed by a hardware scope. At present, we do not support real-time downloading of scopes or the RTW external mode. Et the end of the execution, the last 200 data samples have been stored back to Matlab as a single precision matrix.

If scopes or outputs have been set-up in the scope (see fig. 2) to be stored in the workspace a *add.mat* file will be created. Load the file matrix data into MATLAB with the command:

```
>> load add
>> whos
```

(matrix created by RTW will have the prefix rt_. rt_ScopeData in our case.

```
>> stairs(rt_ScopeData(:,2))
```

The function stairs() is used preferably to plot() to display the results because it doesn't try to smooth the result to display a continues curve. The system is a discrete one and so are the outputs.

Implementation Details and Notes

Do not save too many rows from scopes or 'to workspace' blocks:
- it takes a lot of memory
- the serial link is sometimes shaky, and the bigger the logged data to transfer the more likely the communication may go berserk.
- logging the data takes time and slows down the execution of the model.

Do not forget that the only format supported is the Matix V 2-0-2-2 one. ‘Structure’ or ‘structure with time’ should not be used for data logging. First, structures use up more memory than matrix and memory is scarce on the IDA board. It also allowed me to suppress some code making the application several Kbytes smaller.

The protocol I have implemented for host communications makes the system more sensitive to RS232 line errors as it support different host I/O types (not just echo to a terminal) and the commands must have a precise shape and length. If a spurious byte appears on the line it will be interpreted and things will go wrong. There are no error checking or hand shacking so the host has no way to know if it gets the right information or not.

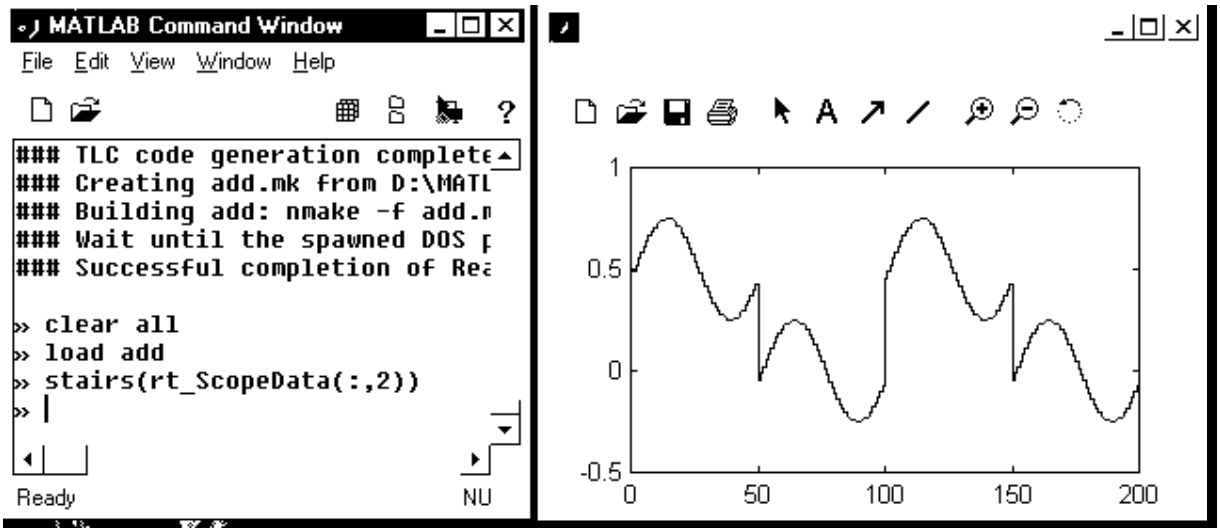

Fig. 4 Presents the Matlab window and the result of the execution on the IDA hardware. Figure 1 plots the last 200 samples of total 200k samples, as downloaded via the seriál line from the IDA board, after the 20 sec. execution run.

Naturally, the C31 floating point arithmetic does not generate 100% identical results to the Simulink double precision. This can be investigated for our case.

```
>> stairs(1000*(double(rt_ScopeData(:,2))-ScopeData(:,2)));
```

`rt_ScopeData` is converted to double, because IDA returns results in a single-precision format to maximise the speed of uploading. See fig. 5.

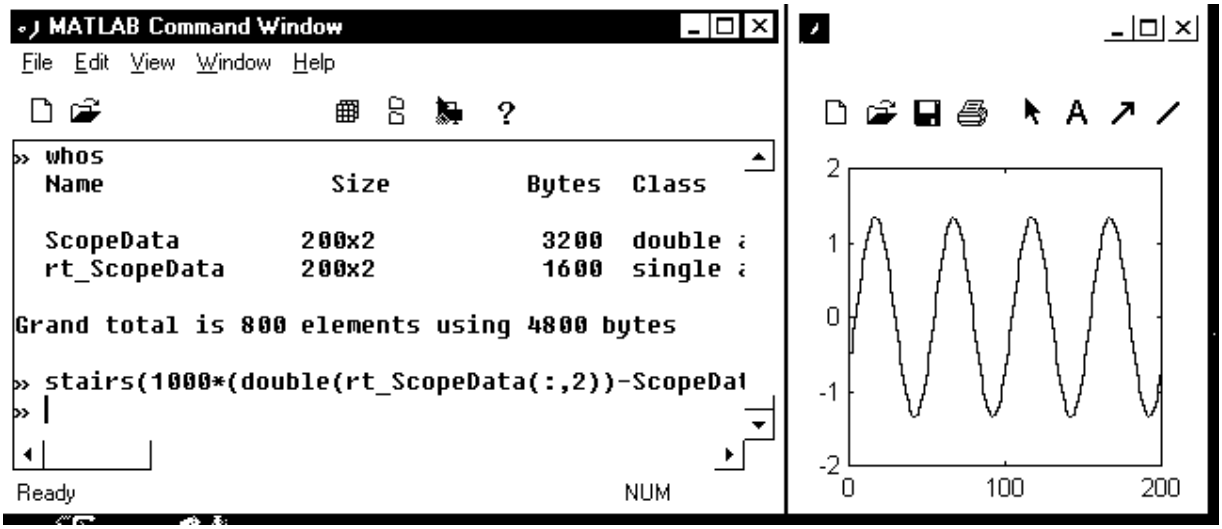

Fig. 5 Difference of the Matlab and the IDA results after 200K samples Notice that the difference is multiplied by 1000 to emphasize the differnce.

If too much data is to be logged (to be later stored into a .mat file), the malloc() function will fail and an error reported. RTW models are memory hungry and there may not be enough RAM available on the IDA board. Unfortunately, the malloc() for logging purpose are made during the initialisation phase and may succeed, while the application will find itself short of memory at a later stage and an implicit allocation will return a wrong pointer without detecting it and the application will die.

grt_c31_notimer.tmf can be used to work out the maximum speed for computing a given number of steps. This may help to find the fastest sampling rate supported by a model running on the IDA board. Be aware that it is an average timing and for some models (cf. RLS) the biggest computation load are not executed for each step. To help finding the impact of multi-rate models, the 'no timer' mode displays the timing for the slowest and fastest loops.

In infinite execution mode (final time == 0) pressing a key is required to stop the execution of the model. If the step size is very small (near the overrun limit) an 'ISR overrun…' error message may come up, it means that it wasn't possible to disable the timer interrupt before a new one occurs. The message should just be ignored.

S-Function design details

The RTW support accepts blocks designed as the TLC code or the classical C-coded S-functions. There are just 2 places where the user should edit the simple Matlab S-function (timestwo.c) to get the first elementary I/O S-function for the IDA board. The ida.h header is needed to declare the IDA macros like the I/O functions:

```
#ifdef  RT
#include "ida.h"
#endif
```

The function `mdlOutputs` takes the input signal and connect directly to the output of the block. In addition it Does the conversion of the input data from the range <-1,1> to the <0,255> range used by the IDA D/A output function `conv_out`.

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T            *y    = ssGetOutputPortRealSignal(S,0);

      *y = *uPtrs[0];

#ifdef  RT
      conv_out((unsigned)(127 * (1.0 + *uPtrs[0])), DACA);
#endif
}
```

The RT directive is defined only for the C31 compilation. That is why the code works fine if compiled for the Simulink by the standard command:

```
>> mex s_out.c
 >>
```

Conclusion

The IDA board is designed and produced by the S.E.E.R. Paris, which has string link to the Czech and Slovak manufacturing. The board could serve as a valid tool for education of the DSP, Control and rapid prototyping concepts in our (CZ, SK … universities and secondary schools). Our attempt is to create reasonable, cost effective option to the existing established solutions.

Contact

UTIA AV CR, (2) Prague 8, Pod vodarenskou vezi 4, 182 08,
e-mail: kadlec@utia.cas.cz, cv@utia.cas.cz, matousek@utia.cas.cz

Grants

This work was partially supported by the EU Esprit LTR project No. 33544 ”HSLA” and grants of the Ministry of Education of the Czech Republic OK314-99 and OK317-99.

References

[1] IDA DSP Development System, User’s Guide. SEER, 49, rue Saint-Didier 75116 Paris tel. (33) 01 45535490, fax. (33) 01 44 05 93 39, www.seer.fr.

[2] IDA Carte DSP. Programmation en C TMS320C3. SEER, 49, rue Saint-Didier 75116 Paris tel. (33) 01 45535490, fax. (33) 01 44 05 93 39, www.seer.fr.

[3] Kadlec J., Vialatte C.: Rapid prototyping and parallel processing under MATLAB 5. In: MATLAB Conference 1997. Kimhua Technology, Seoul 1997, pp. 120-125.