What is the best way to implement my algorithm in Simulink® ?

By Giampiero Campa, PhD, Technical Evangelist
MathWorks, 970 W 190 ST, Suite 530,
Torrance, CA, 90502, USA

giampiero.campa@mathworks.com

Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- Conclusions

An overview of options

- MATLAB® based:
  - MATLAB S-functions
  - MATLAB functions
  - MATLAB System objects™
- C based:
  - C S-functions
  - S-Function Builder
  - Legacy Code Tool (LCT)
- Simulink based:
  - Assembling Simulink blocks

Automatic code (and executable) generation

- No code generation allowed:
  - MATLAB S-functions

- Only toward targets supporting noninlined S-functions:
  - C S-functions

- Code generation allowed toward any target:
  - MATLAB functions
  - MATLAB System Objects
  - Legacy Code Tool
  - S-Function Builder
  - Assembling Simulink blocks

Outline

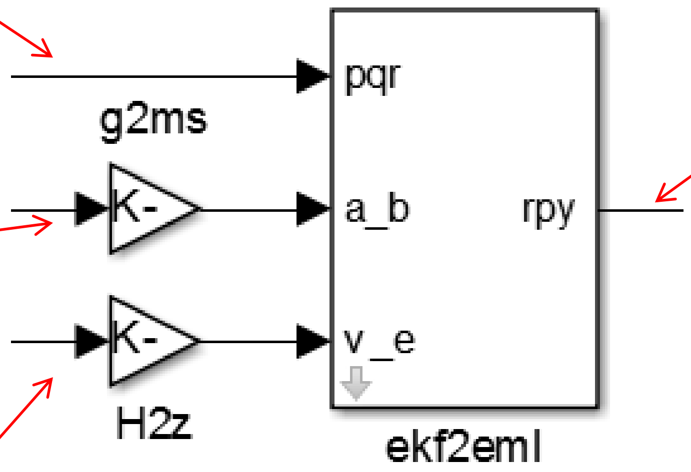
- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- Conclusions

An EKF-based GPS/IMU sensor fusion algorithm for attitude estimation

Angular speed in body frame (from IMU)

Acceleration in body frame (from IMU)

Velocity in earth frame (from GPS)



Roll, Pitch and Yaw angles estimate

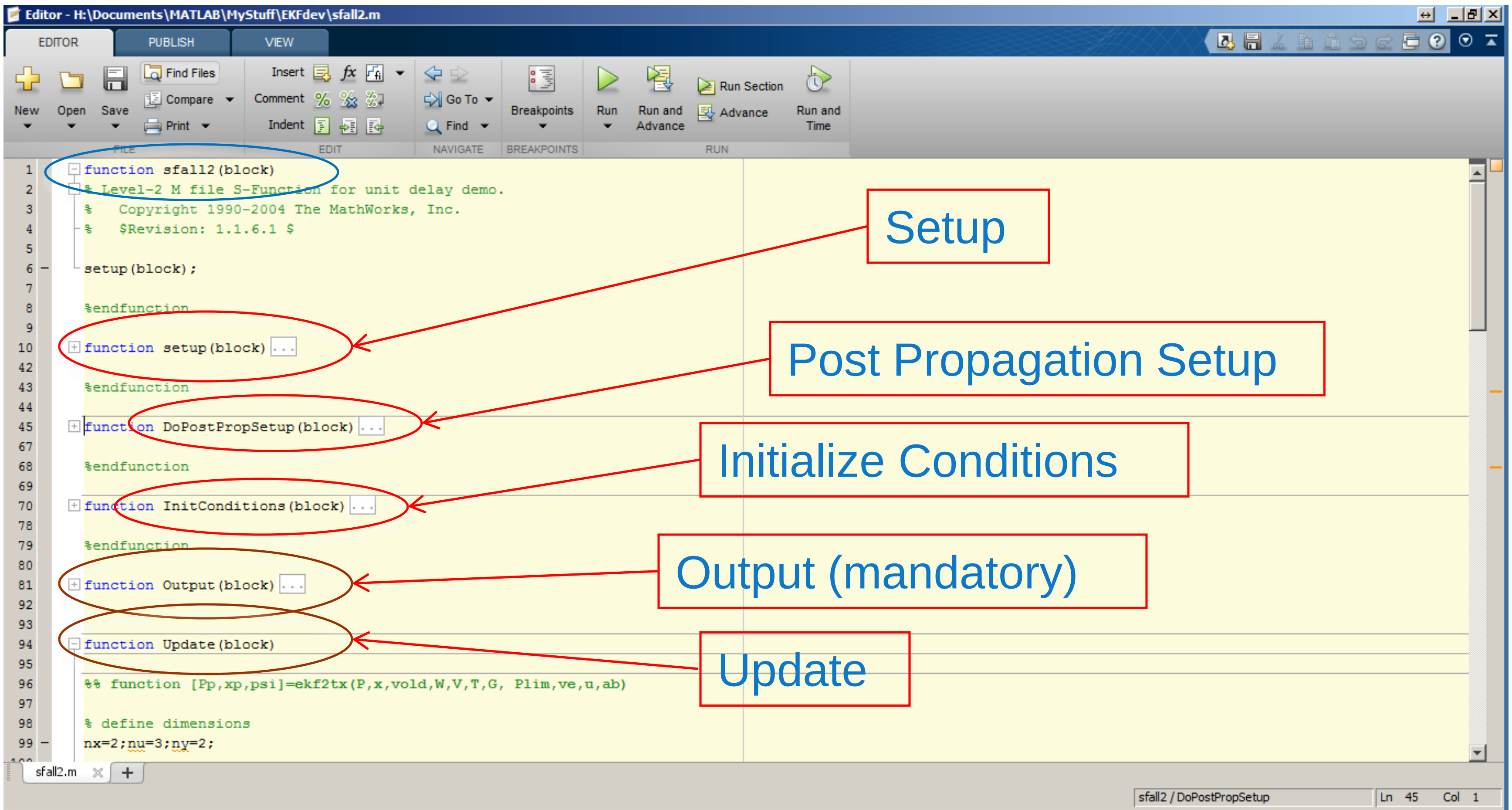
EKF for GPS/IMU sensor fusion: summary

- 3 inputs, each one of size 3x1
- 1 output, also having size 3x1
- Using simplified solution relying only on internal roll and pitch estimates (Kingston-Beard)
  – Internal states are: roll and pitch estimates, a 2x2 P matrix, and the previous velocity in body frame (3x1)
  – Only minor linear algebra required (few 2x2 matrix multiplications and one inversion), so manual coding in C is affordable
- So how do we implement this ?

Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- Conclusions

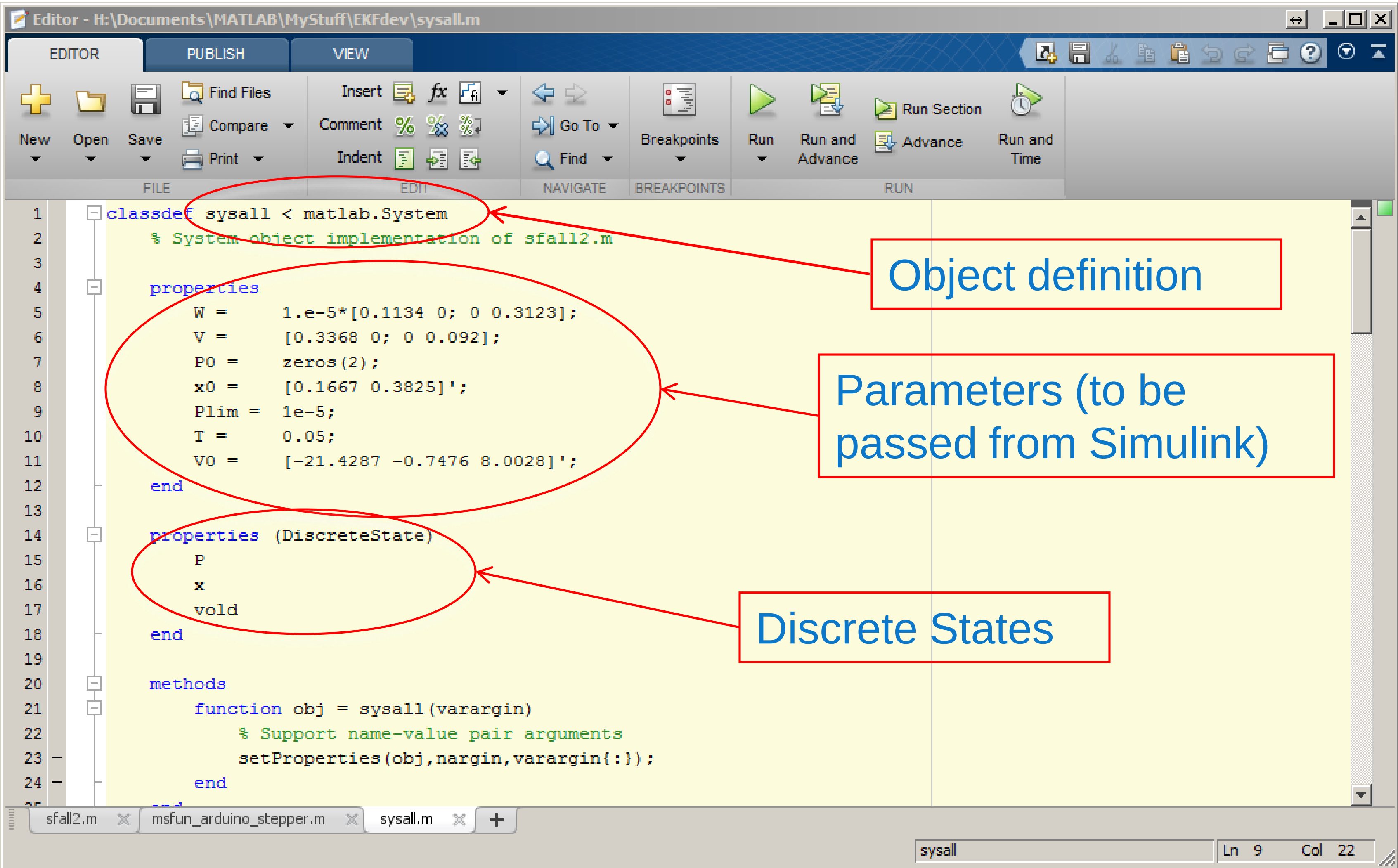
MATLAB S-function block (level 2)



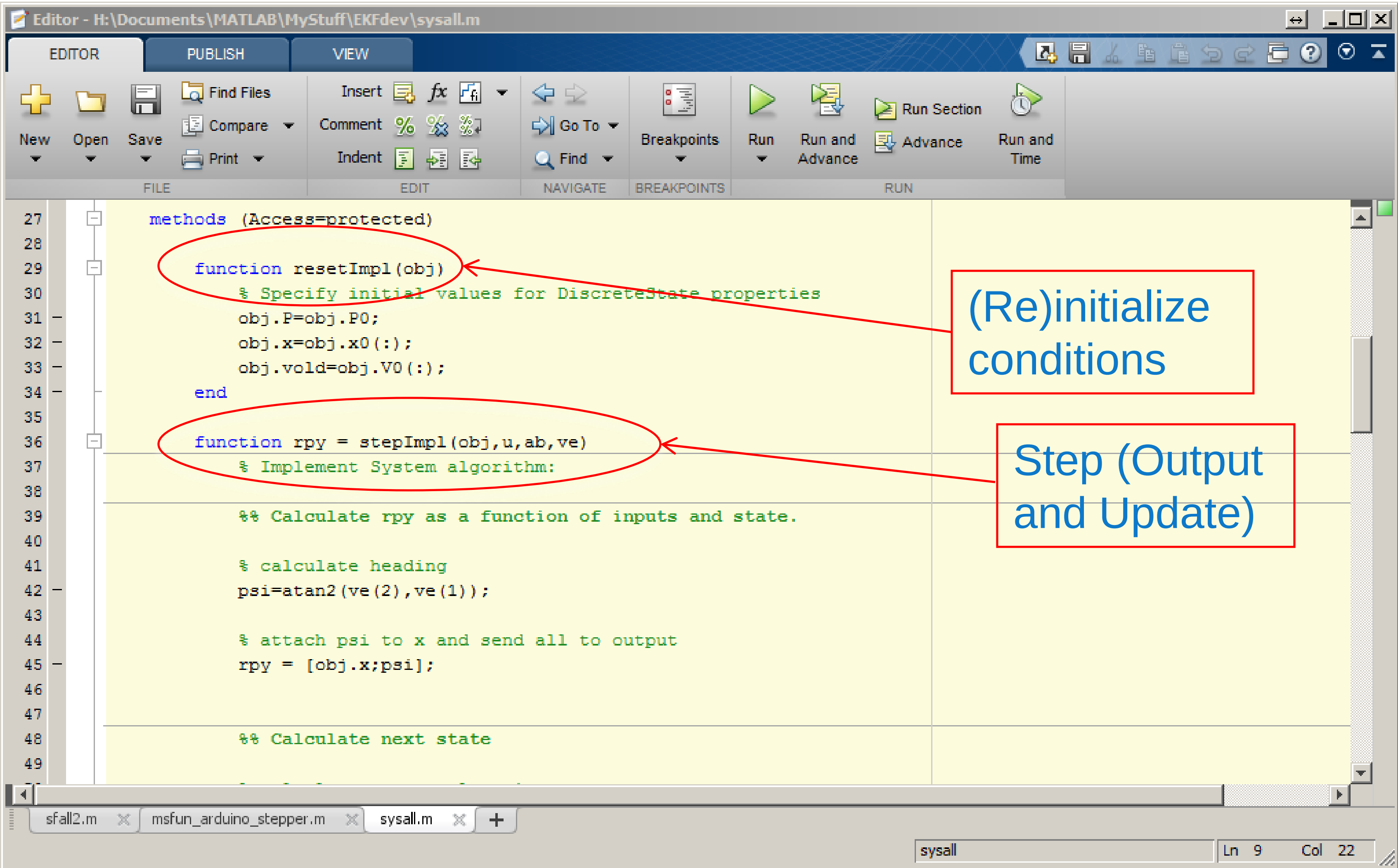
MATLAB S-function block: pros and cons

- Allows **fine control** of sizes, inputs, outputs, states, work vectors, etc.
- Allows use of any MATLAB function, toolbox, or data structure (with few limitations).
- Is **interpreted** (may be slower).
- Does **not allow code generation** and targeting (may only be used for simulation).

MATLAB System object block



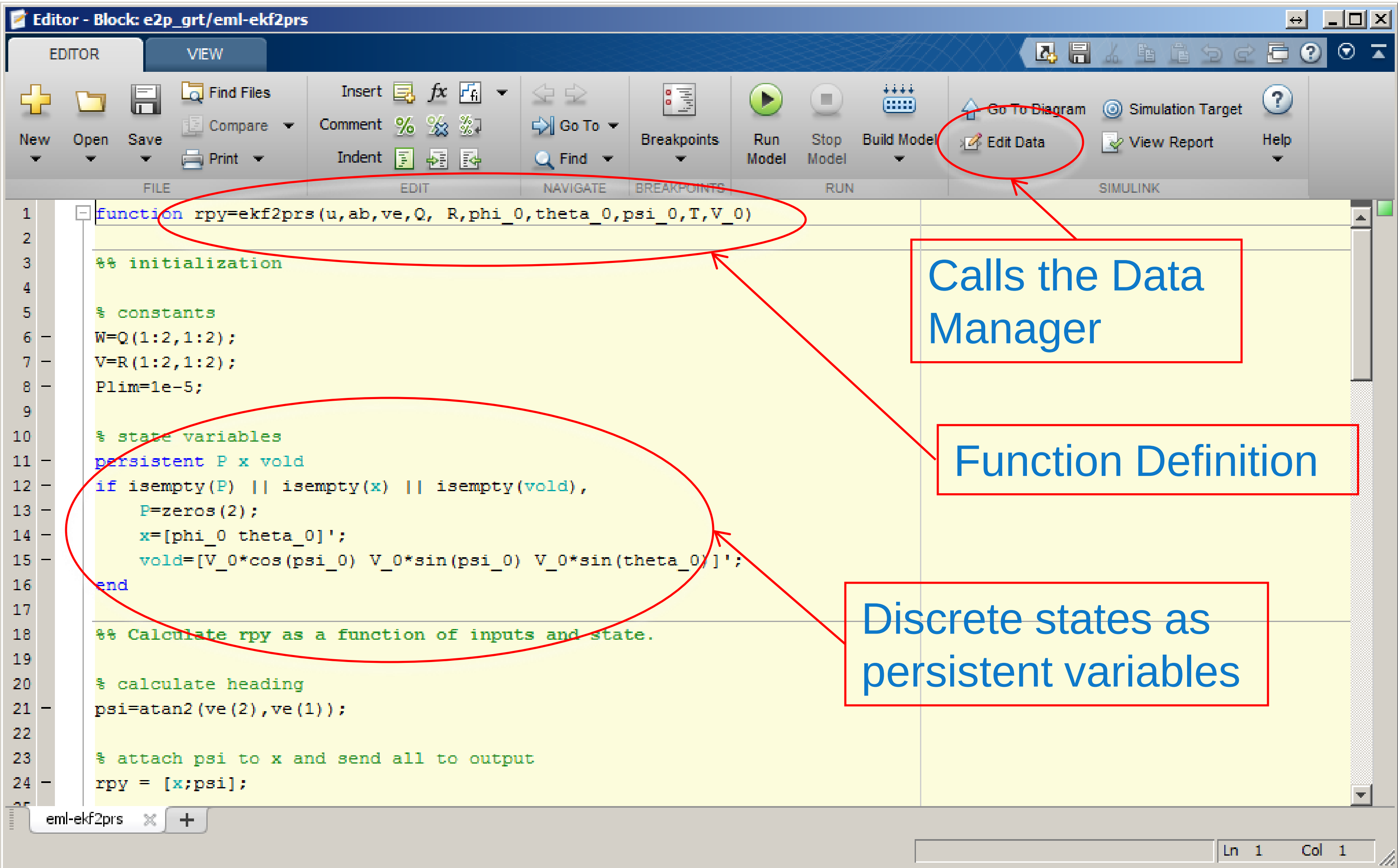
MATLAB System object block



MATLAB System object block: pros and cons

- The API is **simpler and more elegant** than S-functions.
- Allows **code generation** (in simulations and can be executed both in interpreted or compiled mode).
- The mask is generated automatically. Allows for discrete state properties, and it's easier to work with external libraries.
- Allows for **MATLAB-only** (no Simulink) simulations!
- Relies more heavily on OO concepts. Constrained structure may feel too rigid for "one offs", but is good for systematic development, deployment and maintenance.

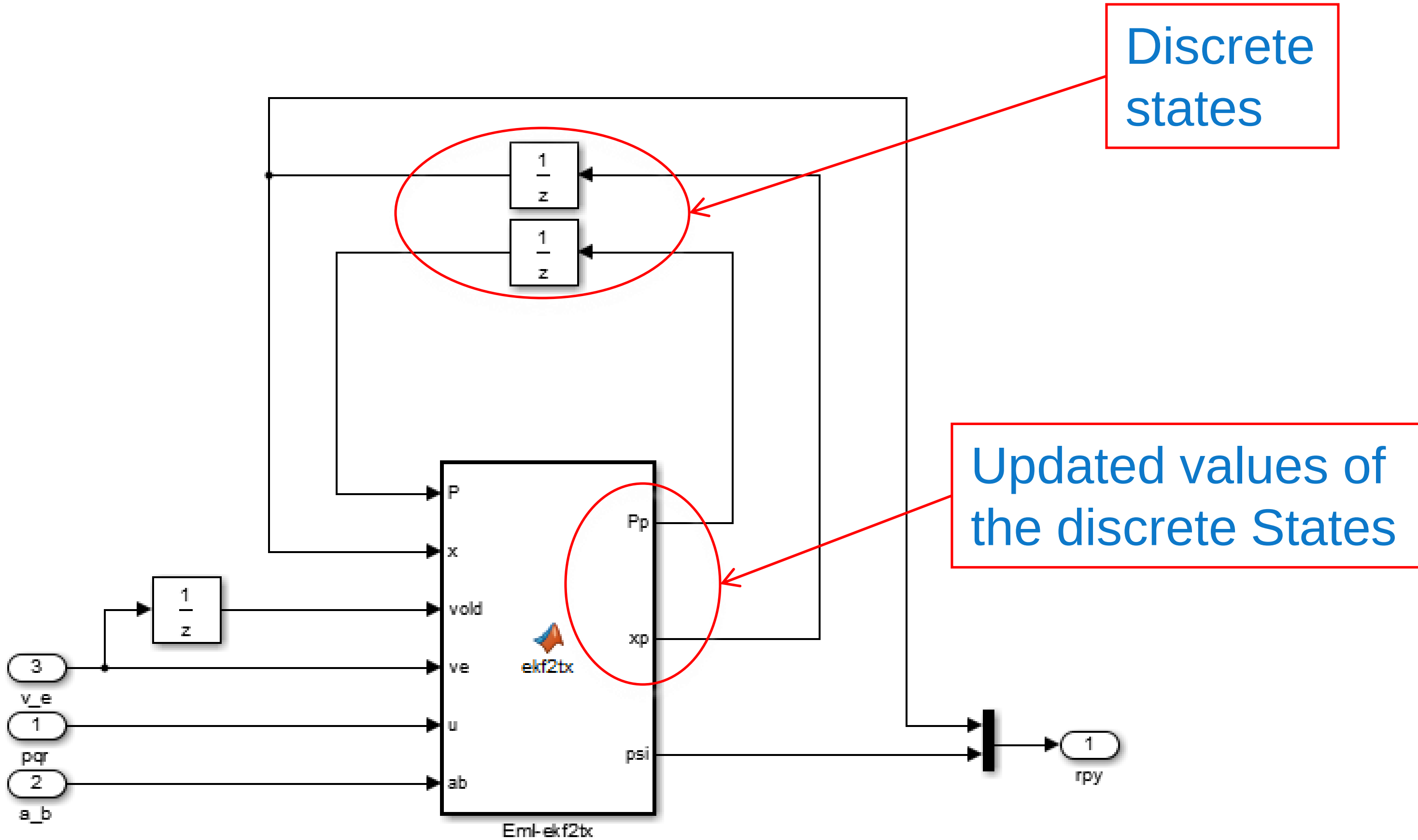
MATLAB function block



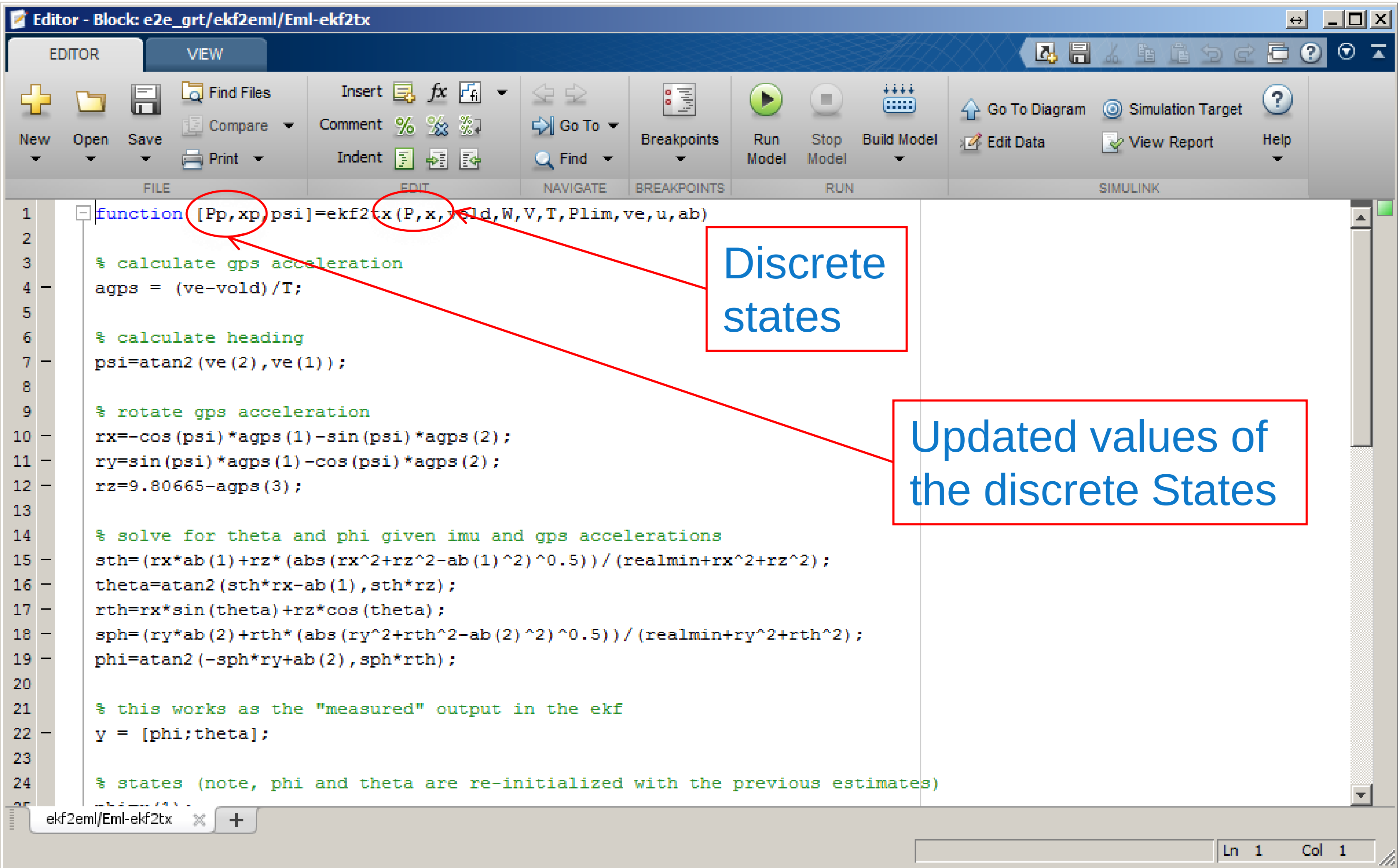
MATLAB function block: pros and cons

- Perhaps the **simplest** method once you know how to use the Data Manager.
- Allows **code generation**
- The default mask is not too descriptive, so a more descriptive mask must be manually added if needed.
- The lack of structure allows for a lot of flexibility and potentially simplifies things.
- Good for “one-off” implementations.

MATLAB function with external states



MATLAB function with external states



MATLAB function with external states: pros and cons

- Is a more structured way of implementing the algorithm in which the states are externally held by unit delays and therefore clearly visible. This simplifies the MATLAB Function.

- Is only here for comparison purposes, probably not worth the extra work with respect to the previous method.

- However it might be useful to implement **continuous time** algorithms. This can be done by using integrators instead of unit delays and calculating (in the MATLAB function block) the state derivative instead of the state update.

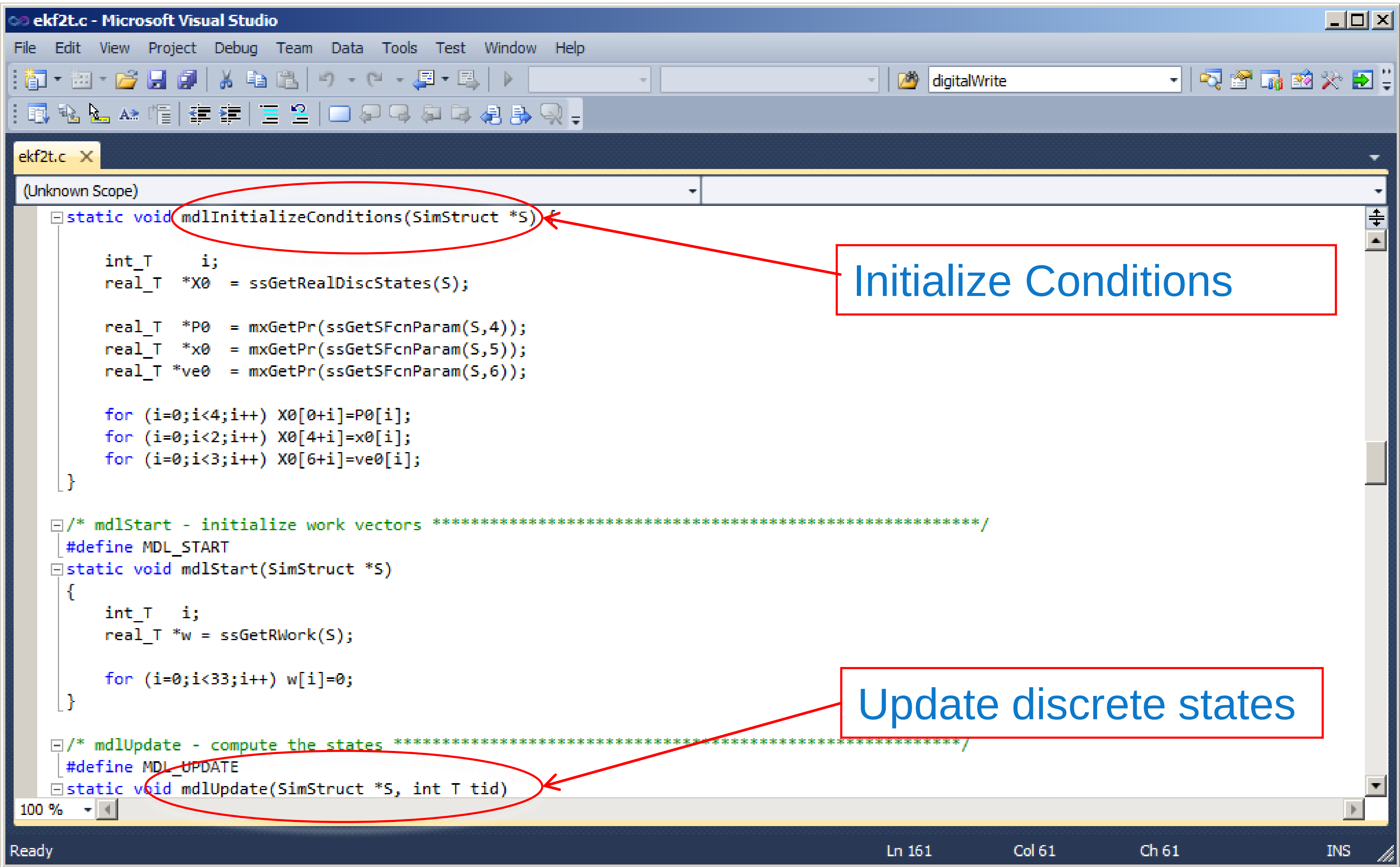
C S-function (level 2)

```c
/*  EKF - Full - with states - S-Function */

#define S_FUNCTION_NAME ekf2t
#define S_FUNCTION_LEVEL 2

#define MAX(a,b)  ((a) > (b)  ? (a) : (b))
#define MIN(a,b)  ((a) < (b)  ? (a) : (b))
#include <math.h>

#include "simstruc.h"

/* mdlCheckParameters, check parameters, this routine is called later from mdlInitializeSizes */
#define MDL_CHECK_PARAMETERS
static void mdlCheckParameters(SimStruct *S)
{
    /* Basic check : All parameters must be real positive vectors                    */
    real_T *pr;

    int_T  i, el, nEls;
    for (i = 0; i < 1; i++) {
        if (mxIsEmpty(    ssGetSFcnParam(S,i)) || mxIsSparse(   ssGetSFcnParam(S,i)) ||
            mxIsComplex(  ssGetSFcnParam(S,i)) || !mxIsNumeric( ssGetSFcnParam(S,i))  )
                  { ssSetErrorStatus(S,"Parameters must be real finite vectors"); return; }
        pr   = mxGetPr(ssGetSFcnParam(S,i));
        nEls = mxGetNumberOfElements(ssGetSFcnParam(S,i));
        for (el = 0; el < nEls; el++) {
            if (!mxIsFinite(pr[el]))
```

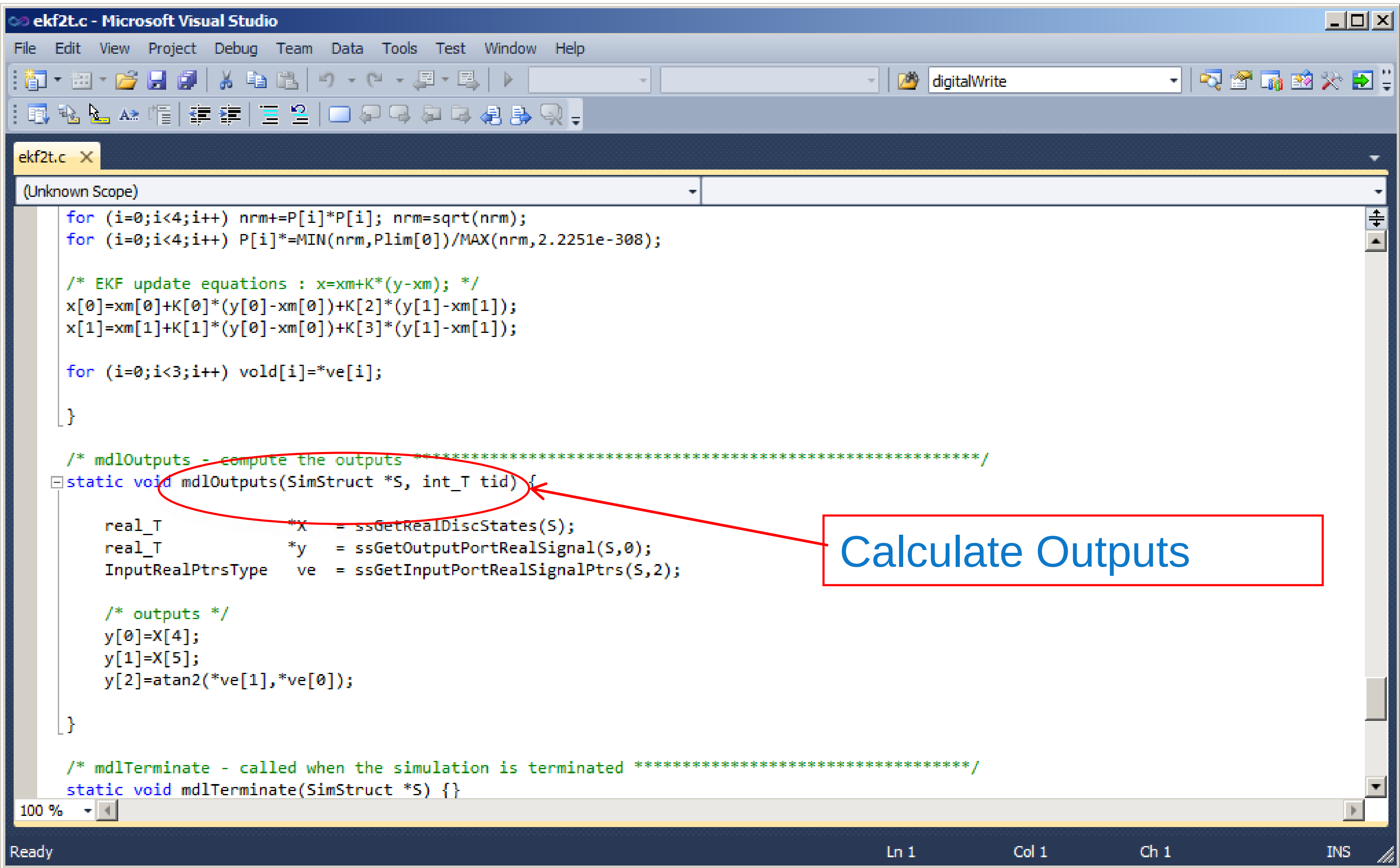
S-function name

Check Parameters

C S-function (level 2)



C S-function (level 2)



C S-function : pros and cons

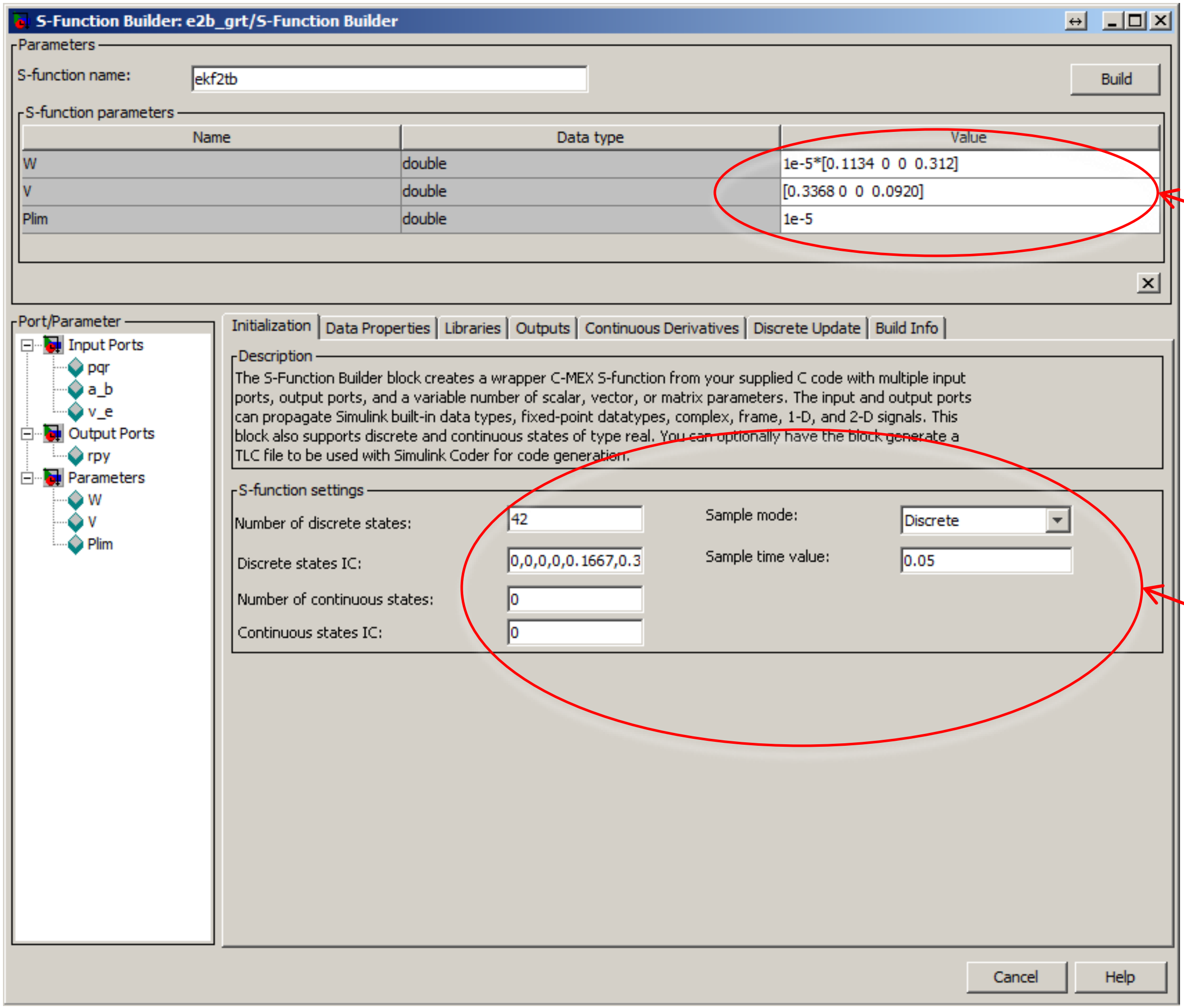
- Supports SimStruct and the entire S-function API (in some sense it is even more powerful than MATLAB S-functions).

- Is **compiled**.

- It must be **handwritten in C** (not feasible for large algorithms requiring linear algebra and/or MATLAB toolboxes functions).

- **Allows code generation only for targets supporting non-inlined S-functions** (unless you write a TLC file).

S-Function Builder

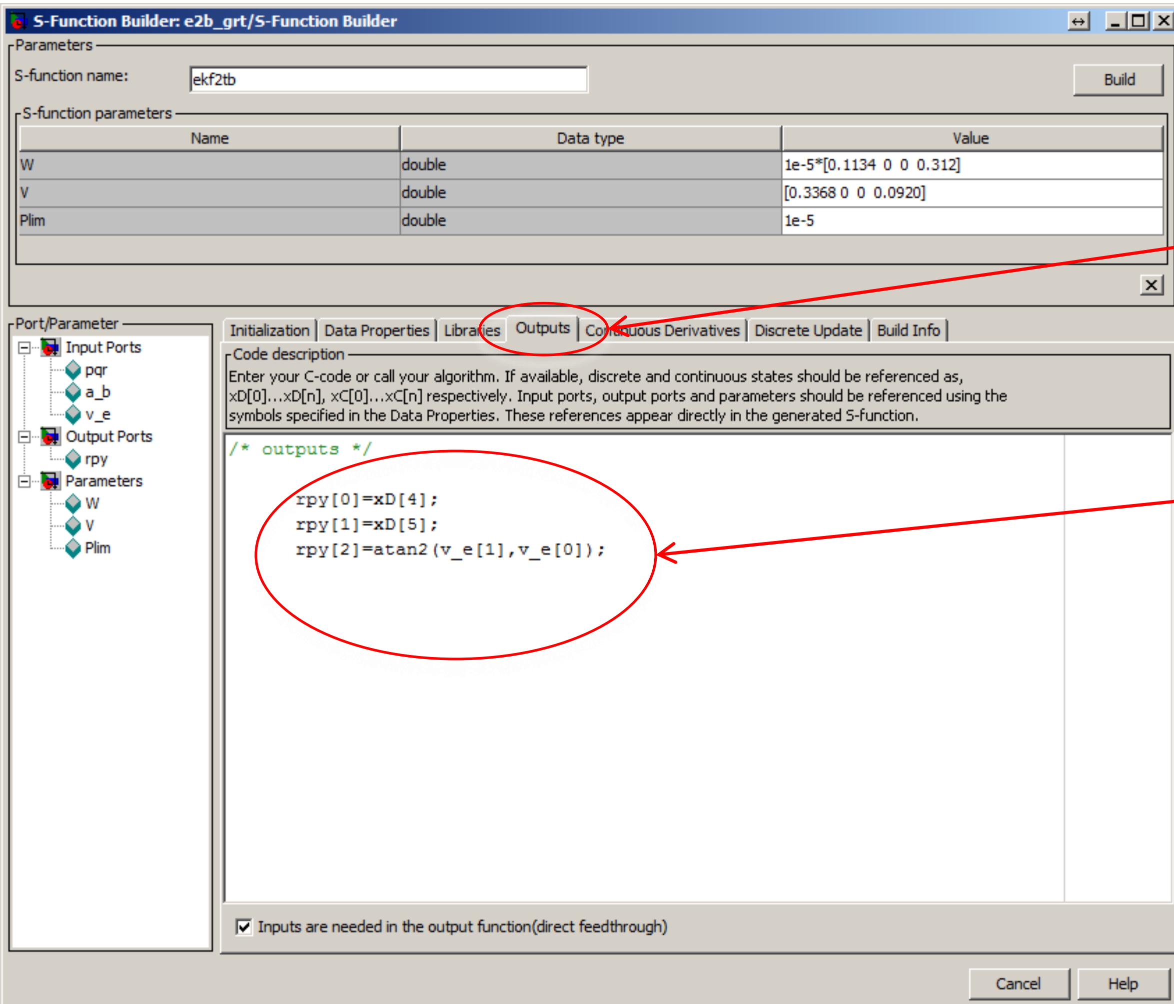


Parameters

Initial Conditions and Sample Time

Must be constants cannot be variables, (rebuild necessary if they are changed).

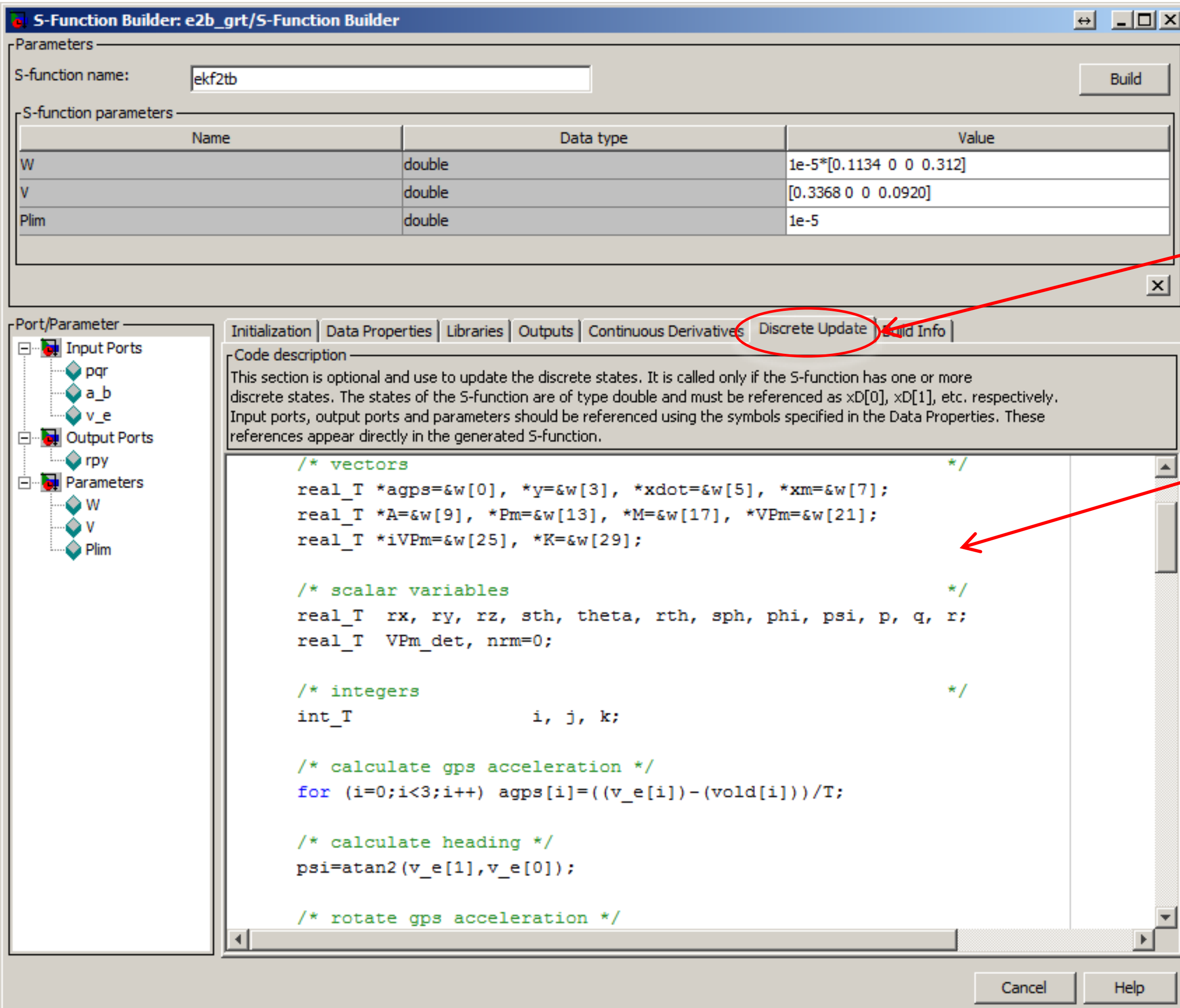
S-Function Builder



Outputs pane

Outputs calculation (xD is the vector of discrete states and work variables)

S-Function Builder



Update pane

Calculation of the update for the discrete states.

S-Function Builder: pros and cons

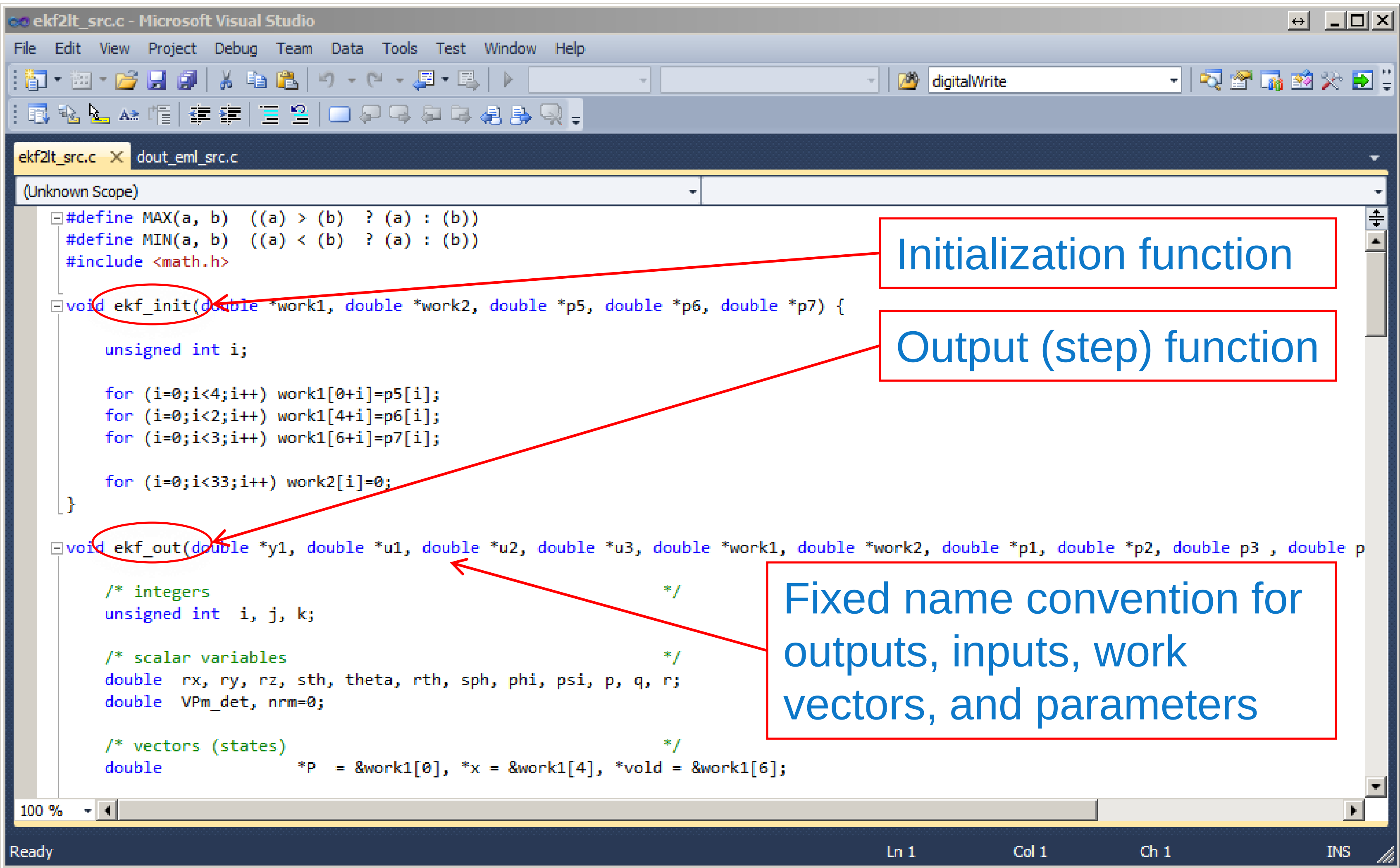
- Less flexible than handwritten S-function. Initial states and sample time **cannot be passed as parameters**. Also, masks are handled differently than other blocks.

- It is compiled. However the generated S-function code uses a wrapper function, which causes a small additional overhead in simulation mode.

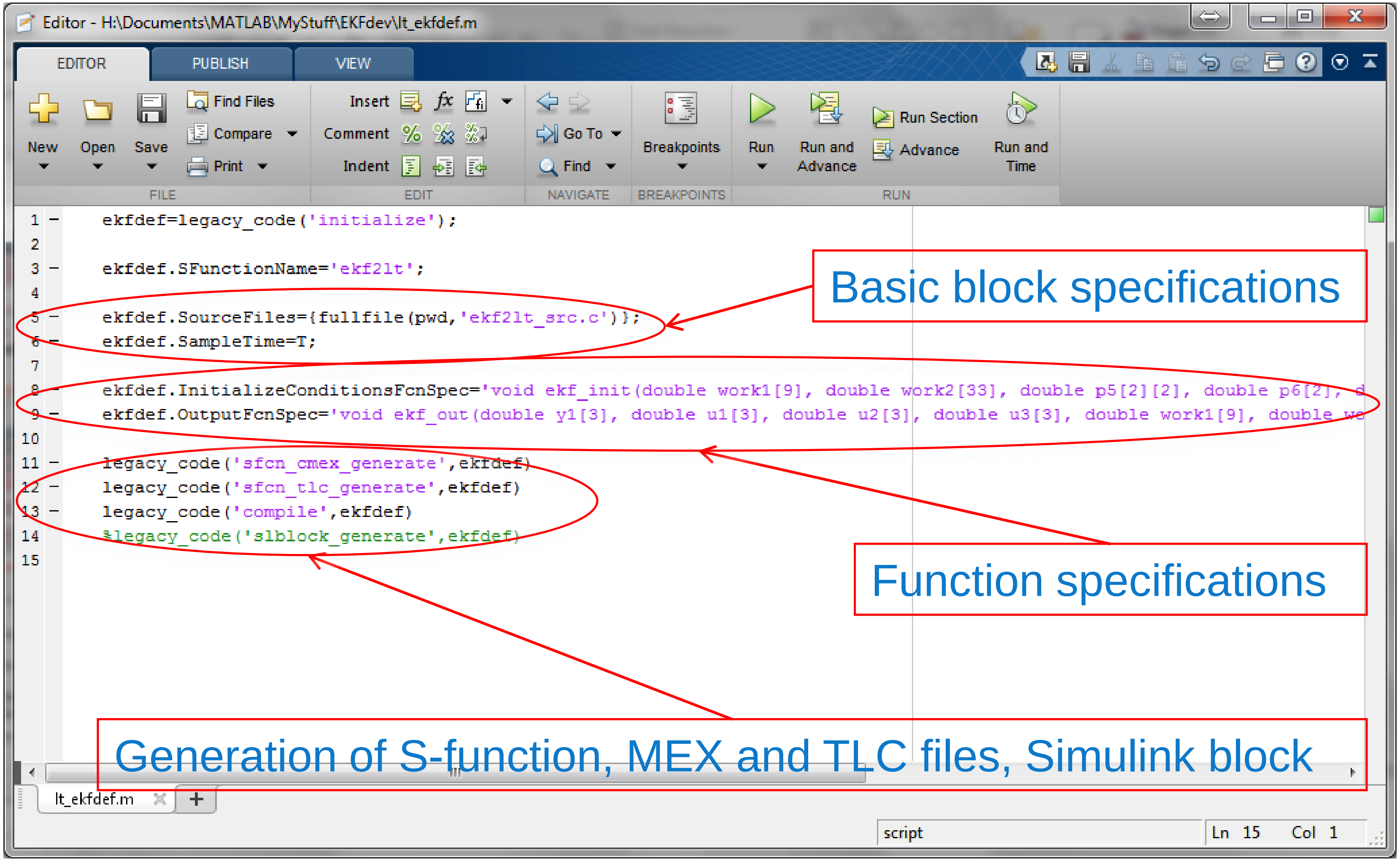
- The builder automatically generates a TLC file, therefore it **allows code generation for any target.**

- It still requires some C and S-function knowledge. Initialization must be performed through update function.

Legacy Code Tool: the C code



Legacy Code Tool: assembling the block



Legacy Code Tool: pros and cons

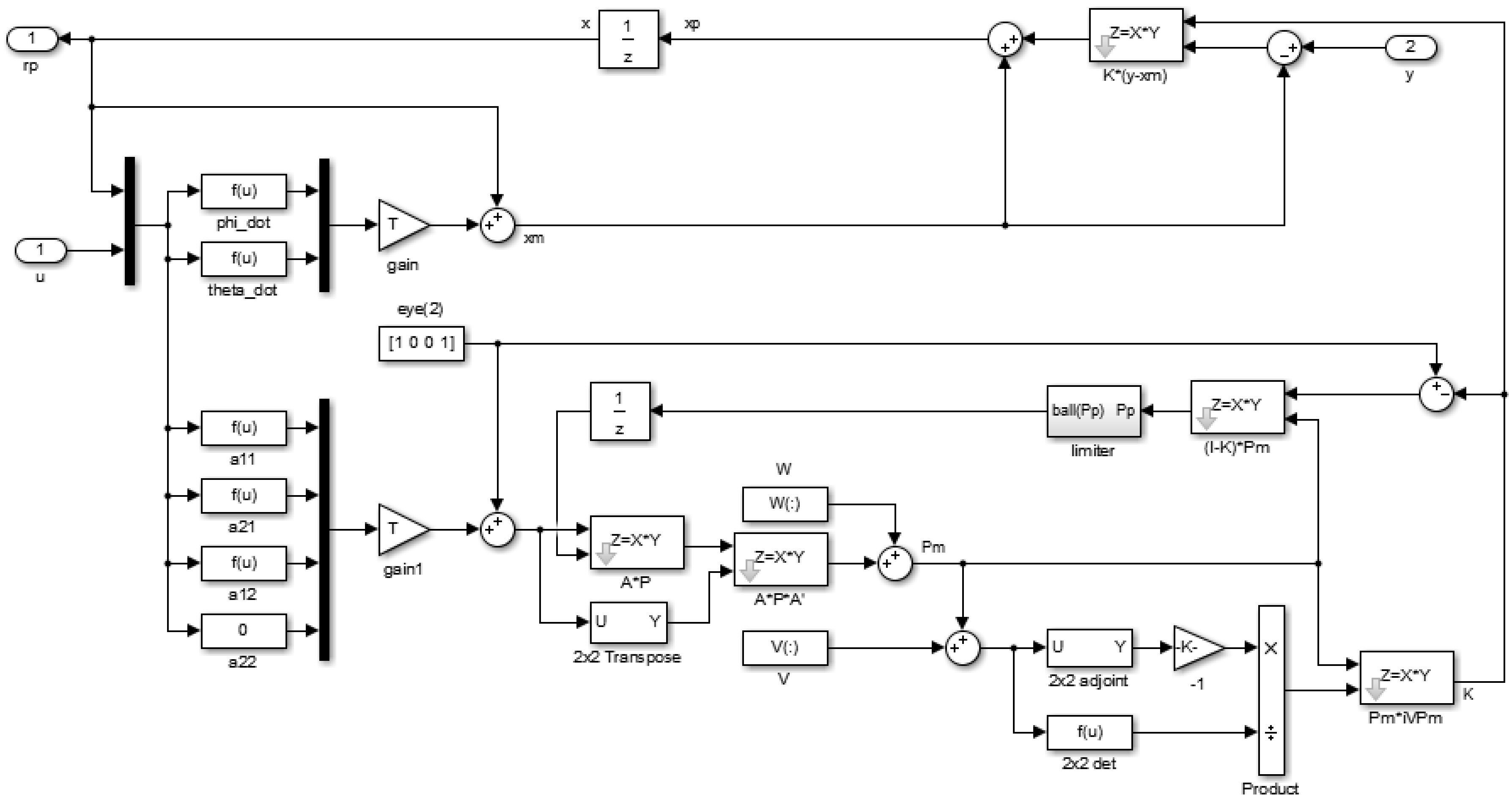
- Completely programmatic interface (no GUI) oriented towards the integration of existing C code.

- It is compiled. It does not use any wrapper. Supports less features than the S-Function Builder.

- S-function and TLC files are automatically generated. **Code generation is allowed for any target** and optimized for faster execution on embedded systems.

- It still requires some C knowledge (but no S-function knowledge).

Pure Simulink



Pure Simulink

- Only Simulink knowledge required.
- It is compiled.
- S-function, MEX and TLC files are not required. Only one model file required thus easy to ship. **Code generation is allowed for any target.**
- Harder for large algorithms requiring either linear algebra, a lot of logic, and/or MATLAB toolbox functions. Harder to deal with the initialization function.

Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- Conclusions

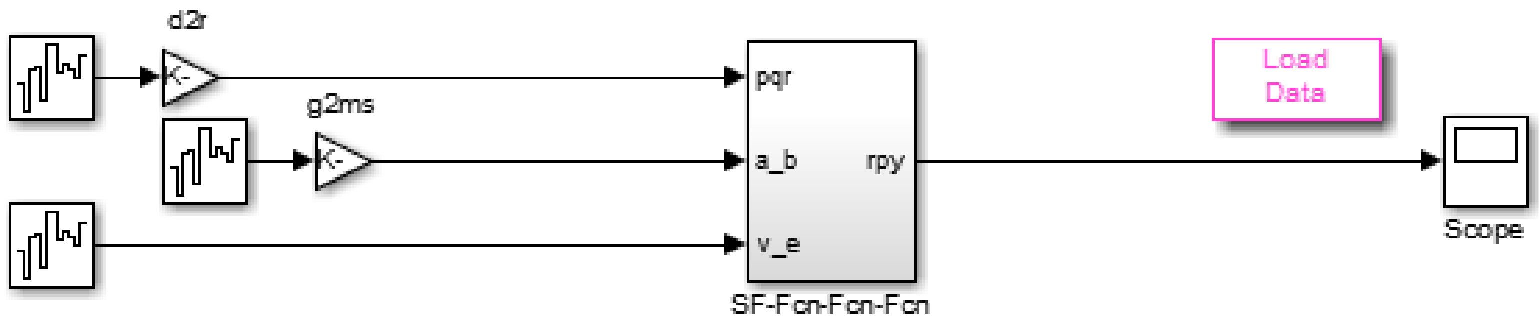
Informal performance comparison

- Simulink blocks were created using the methods previously described (one for each method).

- Simulations were then run to verify that blocks reproduced the same outputs from the same inputs, and starting from the same initial conditions.

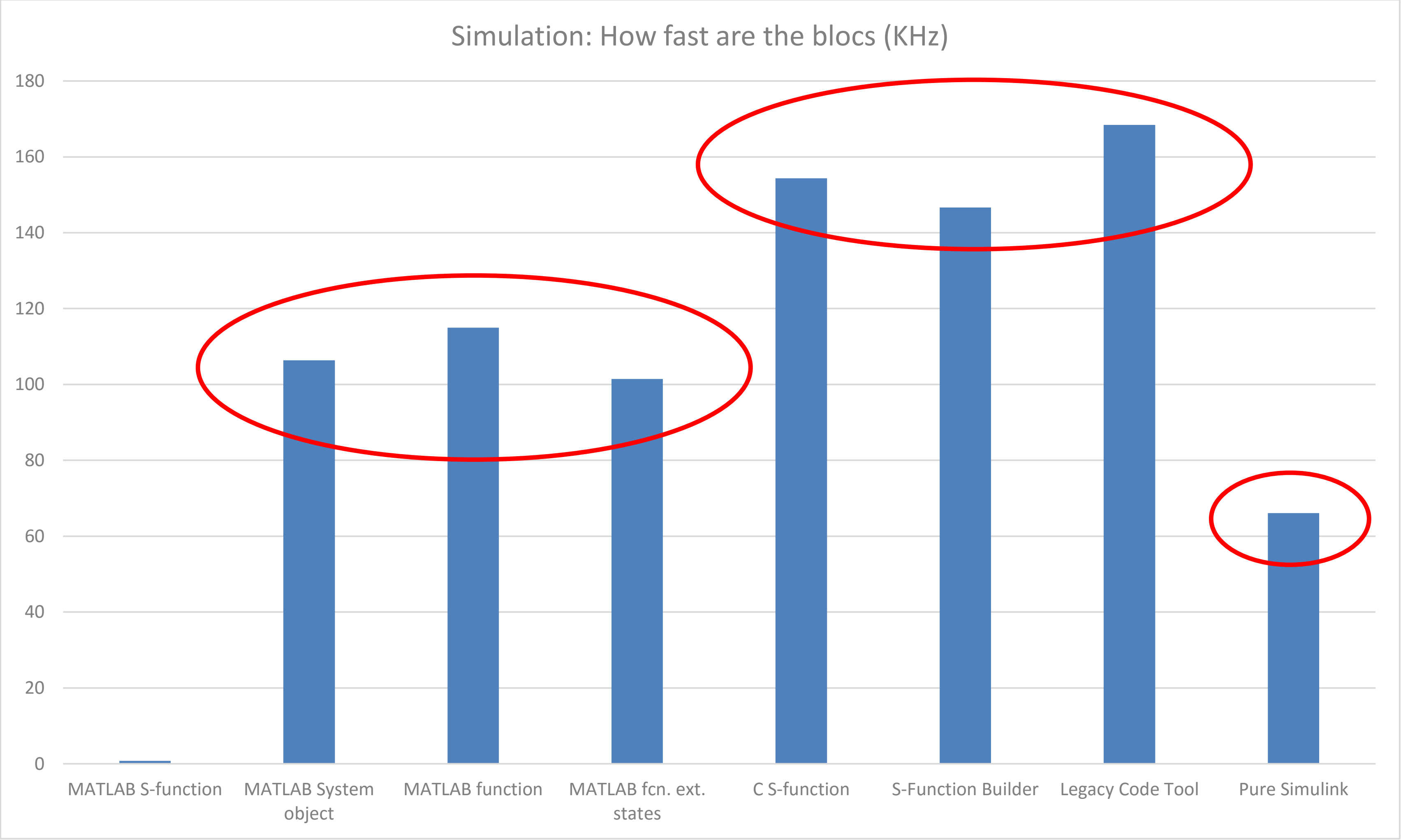
- Simple simulations (containing just a source and the EKF blocks, see next page) were then run programmatically (each one multiple times) in MATLAB 2016b on an Intel Xeon L5630, 2.13GHz, 2-Cores, 8GB RAM, Win7, 64bit laptop.

Example model for performance comparison

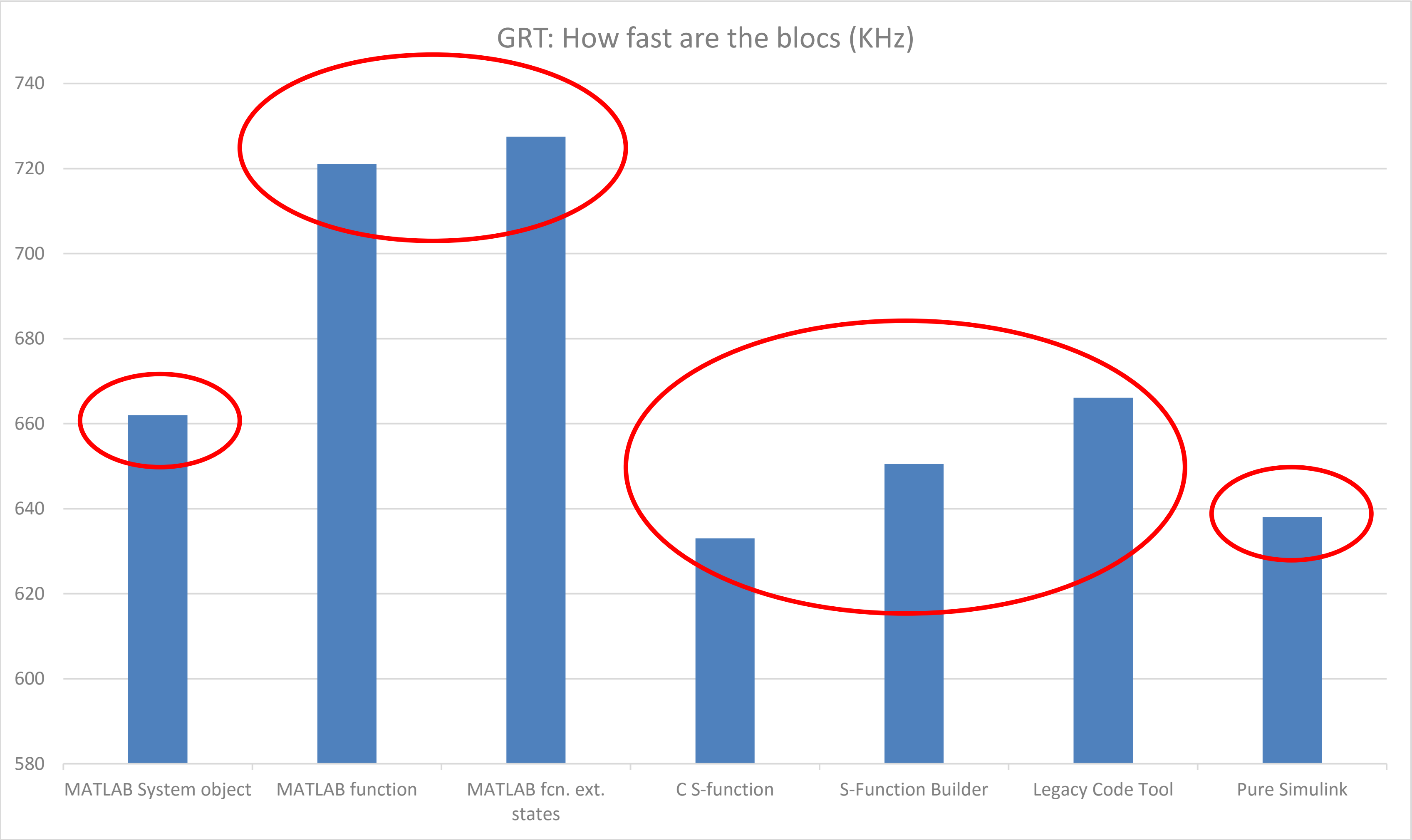


- Simulation time was set to 1e5 seconds, and the sampling time was 0.05 seconds.
- Elapsed time was measured using `tic` and `toc`, and averaged over 4 different executions (so, not rigorous).
- Maximum achievable frequency was calculated dividing the number of steps (1e5/0.05) by the elapsed time.

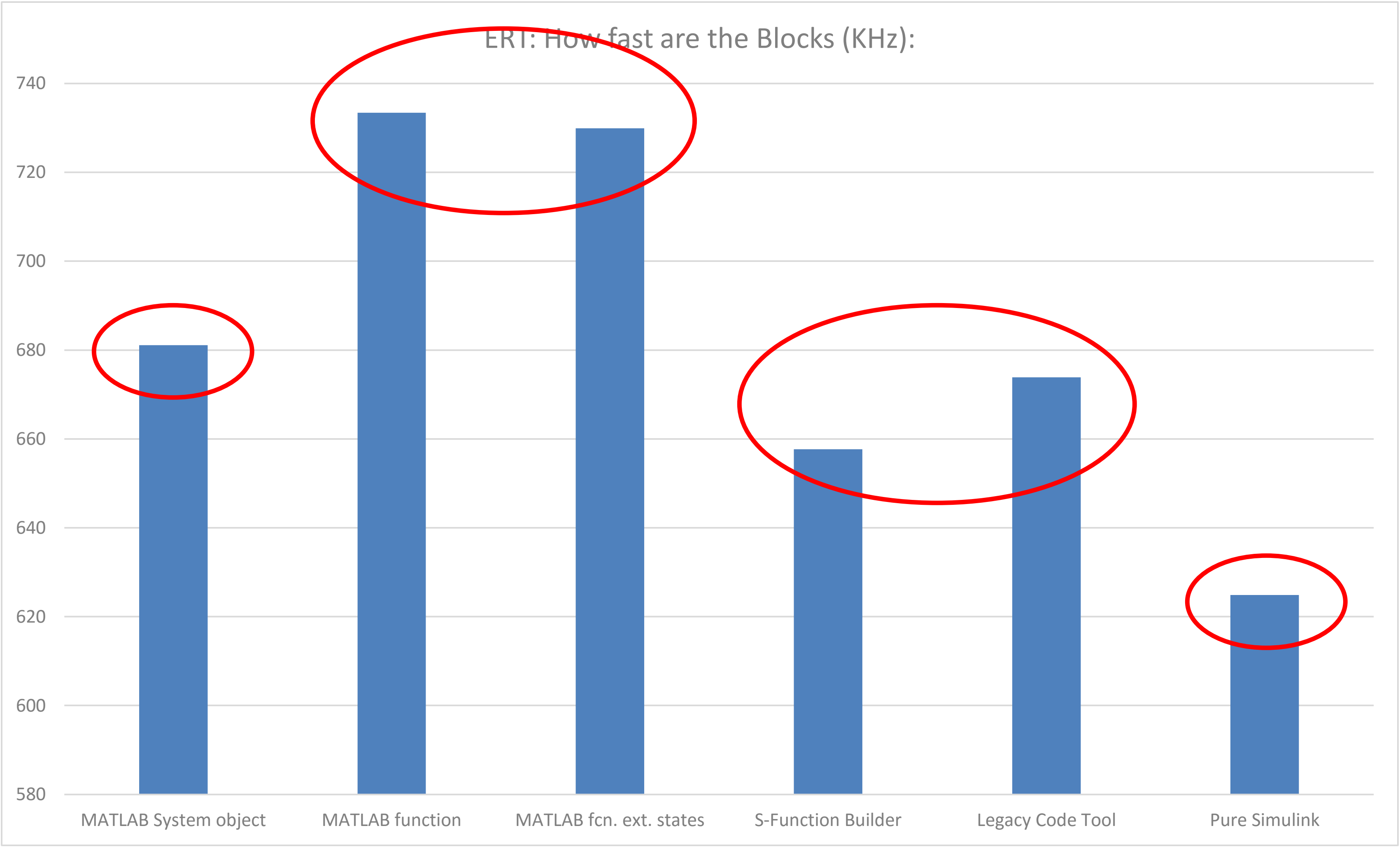
Simulation only



GRT executables

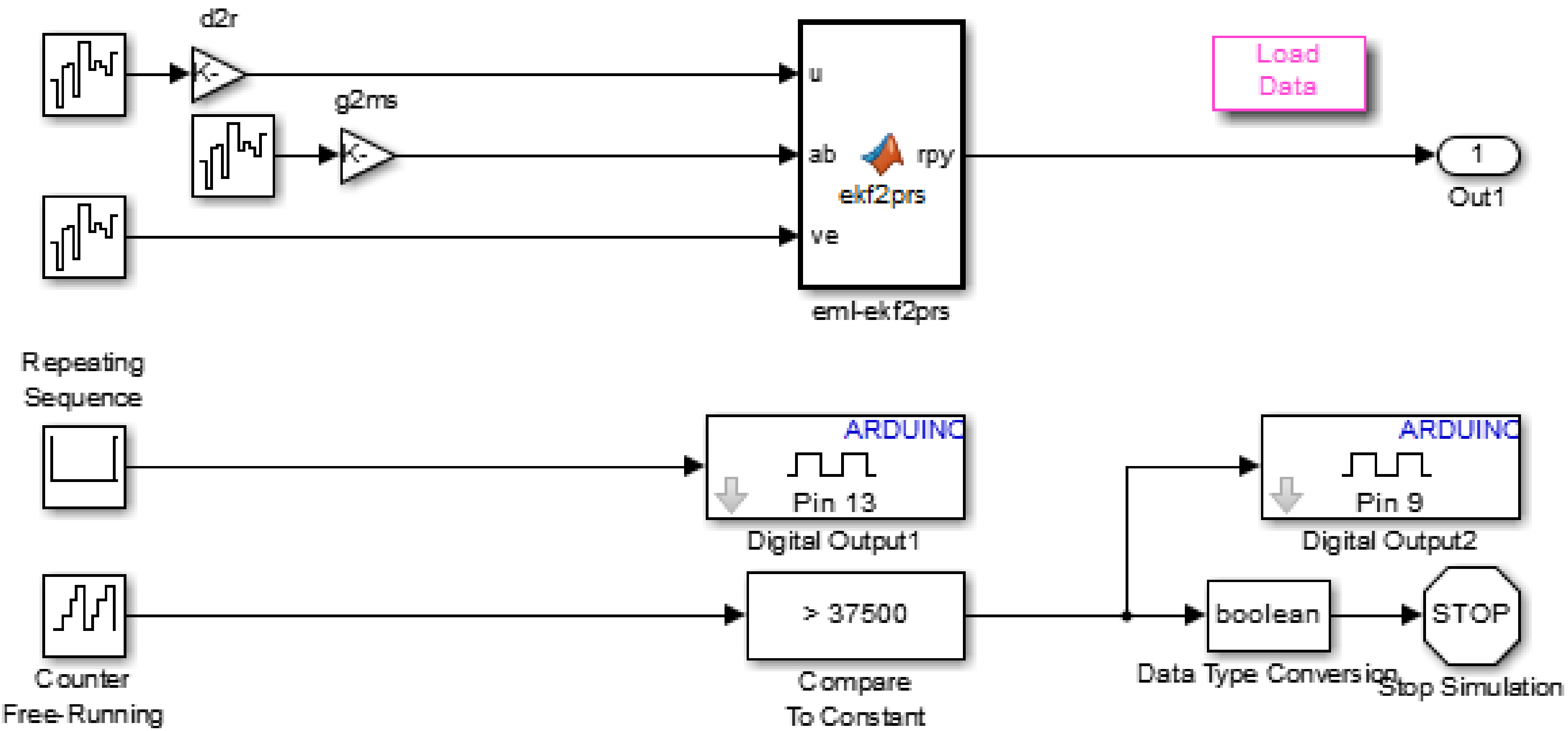


ERT executables



Arduino Uno

The previous Simulink models were augmented with digital output blocks to light up a LED after 5 minutes:



Arduino Uno

- Up until a sampling frequency of 100Hz the execution was fine, and the LED on pin 9 actually lit up after exactly 300s (as measured with a stopwatch).

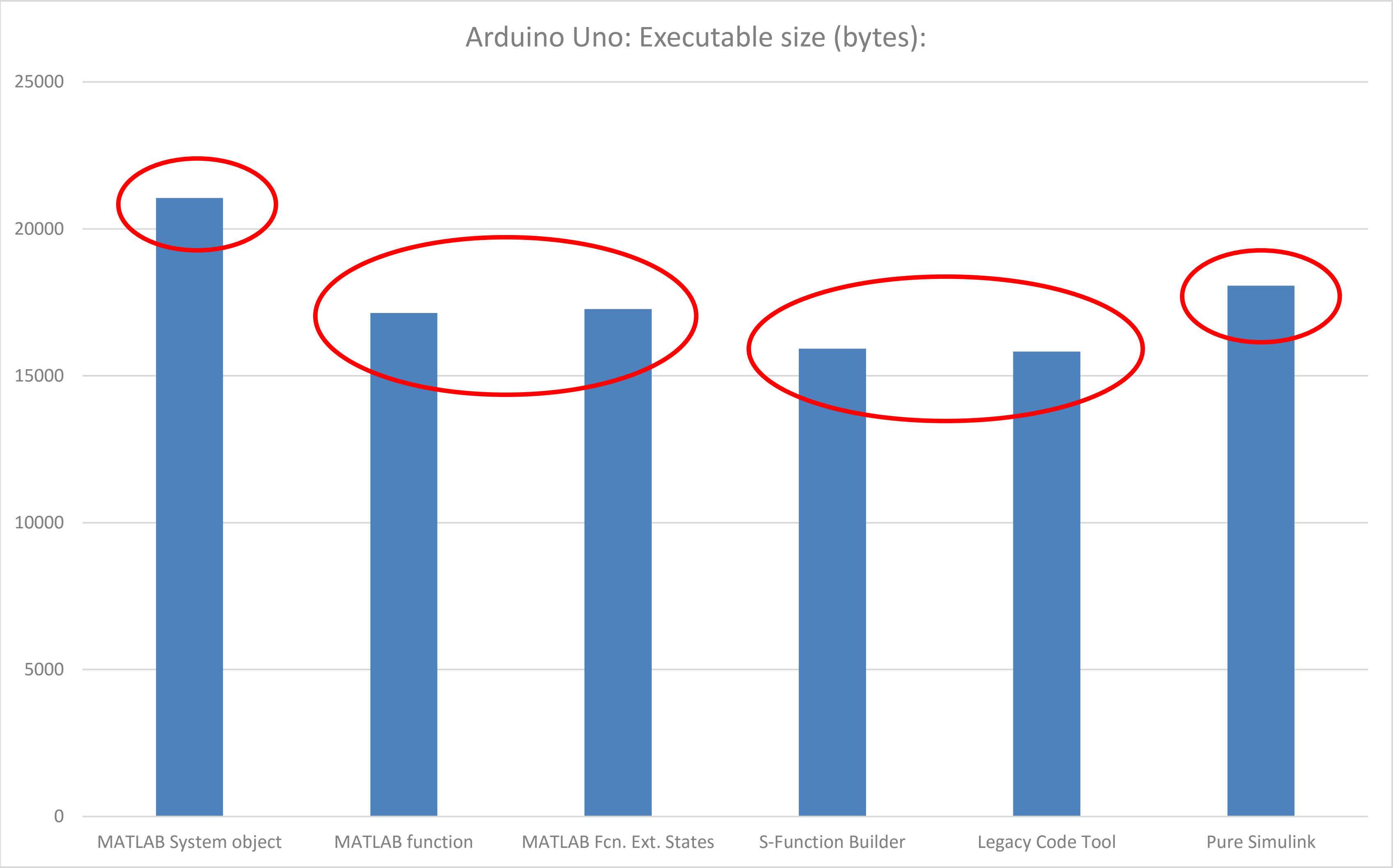
- Whenever the frequency was pushed to 125Hz (base sample time T=0.008 sec) the different executables started to accumulate different delays (so termination happened 30-60s later than 5 minutes).

- Maximum achievable frequency was calculated dividing the number of steps (300/0.008) by the total elapsed time (e.g. 337 sec for the S-Function Builder block)

Arduino Uno

Arduino Uno: How fast are the blocks (Hz):
116
114
112
110
108
106
104
102
100
98
MATLAB System object
MATLAB function
MATLAB Fcn. Ext. States
S-Function Builder
Legacy Code Tool
Pure Simulink

Arduino Uno



Arduino Uno: Executable size (bytes):
25000
20000
15000
10000
5000
0
MATLAB System object
MATLAB function
MATLAB Fcn. Ext. States
S-Function Builder
Legacy Code Tool
Pure Simulink

Outline

- Implementing algorithms in Simulink: overview
- An Extended Kalman Filter (EKF) for GPS/IMU Fusion
- Case Study: Implementing the EKF as a Simulink block
- Informal performance comparison
- Conclusions

Conclusions

- MATLAB System object, MATLAB function, S-Function Builder, Legacy Code Tool and pure Simulink work for any kind of target.

- The performance comparison was somewhat informal (and compiler-dependent), however:
  - **Methods based on C tend to be faster only in simulation**
  - **Methods based on MATLAB tend to be faster for on-target execution.**
  - The MATLAB Function bock is marginally faster than the System Object block, and is OK for one-offs, but the latter has many features that make it easier to develop, deploy and maintain a block.