xPC Target Real-Time Environment for The MIT Cheetah Robot

Zakaria Ahmad
Georgia Institute of Technology

under the direction of
Prof. Sangbae Kim
Biomimetic Robotics Lab
Department of Mechanical Engineering
Massachusetts Institute of Technology

Summer Research Internship
Kuwait-MIT Center for Natural Resources and the Environment

August 30, 2013

Abstract

Using MATLAB/Simulink for model-based design of algorithms and control allows for faster rapid control prototyping (RCP) and reduces errors implementation and syntax errors when compared with source-based design. A second generation robot, which is bio-inspired by a cheetah, is being developed at MIT's Biomimetic Robotics Lab. The cheetah robot uses xPC target as its real-time environment and communicates with its sensor and actuators via serial. This report describes the procedures necessary to setup the real-time environment as well as communicate with the sensor and actuators. In summary, the real-time environment was successfully bench-tested and will allow faster RCP to accelerate the development of the second generation cheetah robot.

1. Introduction

Robotic systems, for many years, have been attractive to applications or operations where human life is at risk and applications which require repetitive tasks. For instance, robots can be sent to inspect hazardous conditions in toxic environments such as nuclear power plants or navigate areas which humans cannot reach such as deep areas of the ocean. These are only two examples of where conventional robots (i.e. robots with four wheels) have failed to reach the desired location due to their dependency on wheels for motion. In the Fukushima Daiichi Nuclear Power Plant in Japan, robots failed their mission to switch off the reactors since the ground was no longer a smooth surface (due to the earthquake) for wheels to roll on. Another incident is the Gulf of Mexico oil spill, where conventional marine robots could not reach the source of the oil spill since the pipes were leaking at a very deep level of the sea.

One way to solve such limitations in robotic systems is to design robots inspired by animals. For instance, a four-legged animal (such as a cheetah) can navigate any surface whether it was rough or smooth. The Biomimetic Robotics Lab at Massachusetts Institute of Technology led by Dr. Sangbae Kim aims to design robots inspired by animals to overcome limitations addressed by conventional robots. A first generation cheetah robot has been successfully designed and is able to reach speeds of 22 kph on a smooth surface. A second generation cheetah robot is being designed to navigate on rough surfaces. For this purpose, additional sensors are required to stabilize the cheetah robot and an environment which enables fast rapid control prototyping (RCP)

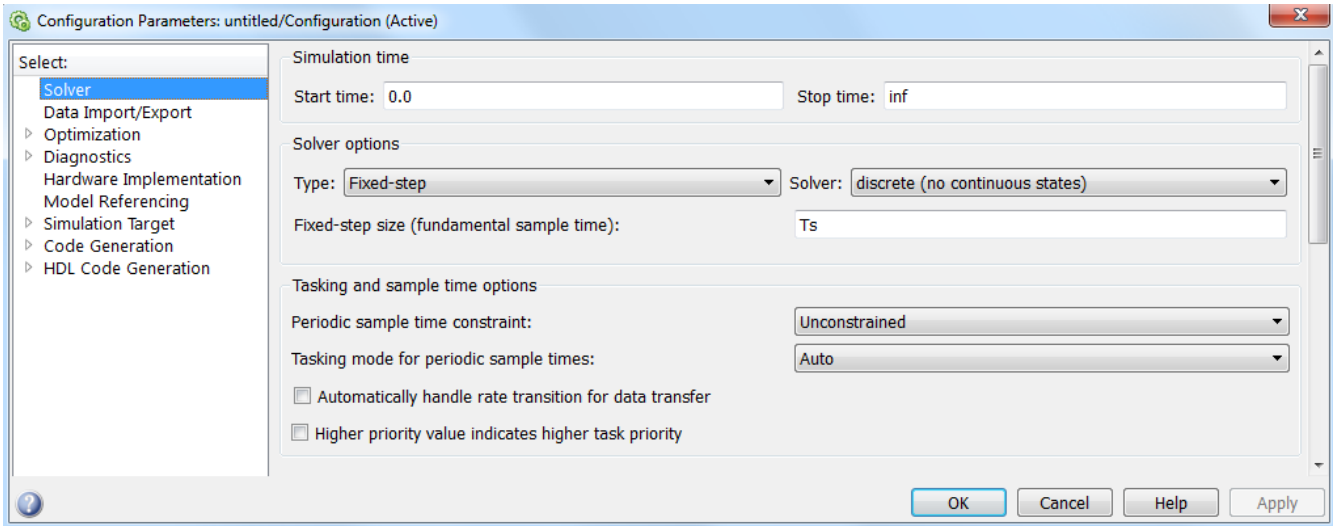
The first generation cheetah robot used LabView and a field programmable gate array (FPGA) to communicate with the sensors and actuators. This hardware and software setup proved its limitations when the time came to rapidly test different algorithms. For this reason, the second generation cheetah robot will use MATLAB/Simulink which allows model-based design for the algorithms and control. The advantage in using model-based design is the designer doesn't have to worry about the implementation of algorithms in source code since the code is automatically generated by the Simulink coder. This reduces the implementation and syntax errors allowing designers to focus on developing the algorithms and rapidly prototyping them.

The aim of this project was to prepare the hardware and software to facilitate model-based design for the cheetah robot in a real-time environment; specifically, the xPC target. The format of this report is such that the reader can replicate what the author accomplished given the reader has the same hardware and software. The report first starts with setting up the xPC target environment then the

communication via serial with the inertial measurement unit, the dynamixel motor, and the motor driver.

2. xPC Target on the Speedgoat

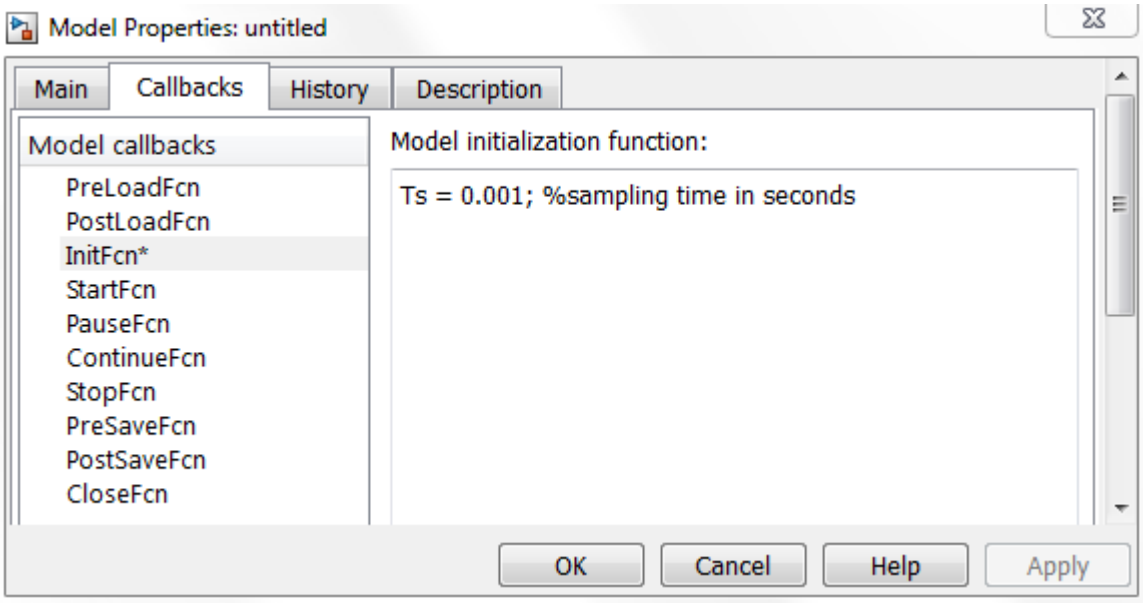
xPC target is a real-time environment solution developed by the Mathworks Inc. and is available in MATLAB/Simulink. The xPC target is actually on the software environment and doesn't impose any restrictions on where it can be installed. A user can create a bootable image and run the xPC target real-time environment on a desktop. However, for the cheetah robot, a Speedgoat computer was purchased and the xPC target was installed on it. First, xPC target version 5.4, which comes with MATLAB 2013a, was used by the author. Next, the following steps need to be modified in Simulink: from the top bar go to Simulation > Model Configuration Parameters, then modify the necessary parameters under Solver to match the following screen. Note: the sampling is specified as a variable, names Ts, and will be assigned later.



On the same window, click on “Code Generation” on the left and click on “Browse”. Select “xpctarget.tlc” from the list. These are the minimum settings to enable Simulink to build models that run on an xPC target.

2.1 Sampling Time

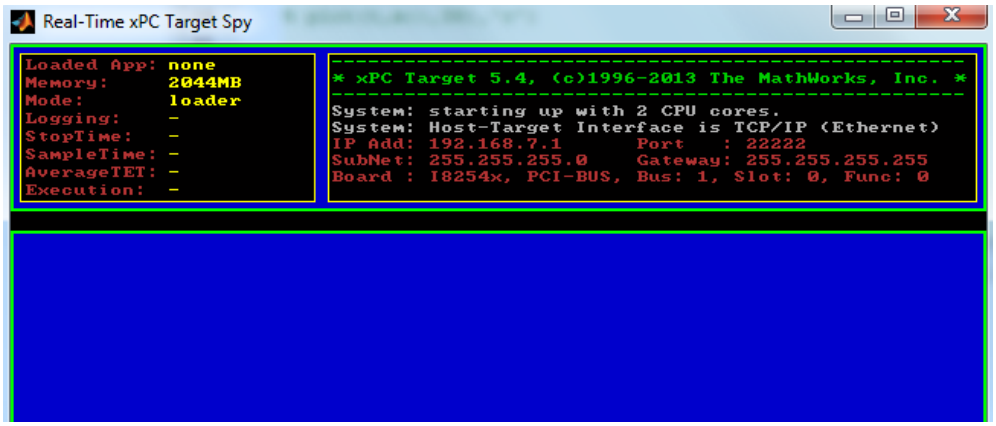
The sample time, Ts, was left as a variable in any block which requires a sampling time (such as as FIFO Read block). This way, if one desires to experiment with different sampling times then Ts can be changed in only one place instead of multiple blocks. The cheetah robot will be running the algorithms at 1 kHz. To set the sample time, go to File > Model Properties > Model Properties. In the “Model Properties” window click on the “Callbacks” tab and then click on the “InitFcn” and type Ts = 0.001 for a 1ms sampling time (or 1 kHz). The “InitFcn” is an initialization function and is called before the Simulink model is executed. By specifying the sampling time in this callback, the variable Ts is evaluated and is written to the MATLAB workspace. The following screen shot shows the correct settings:



2.2 Speedgoat setup

Note: The Speedgoat driver and library need to be installed for the first time. Please refer to the Speegoat documentation for this purpose.

The Speedgoat communicates with the host via an Ethernet connection. It is therefore necessary for the host computer to have a dedicated Ethernet card for the sole purpose of communicating with the Speedgoat. Once the Speedgoat is powered, the IP Address, Port, and Subnet of the host computer should be changed to match the Speedgoat's. The settings are specific to each operating system and it is left to the reader to figure out the necessary steps. The reader can always verify if the settings are correct by pinging the IP Address of the Speedgoat and seeing its response. Once the correct settings are modified, the ***xpcexplr*** command can be entered at the MATLAB command line and the Speedgoat can be added as an xPC target. For adding an xPC target, please consult the xPC target documentation. If the Speedgoat is successfully connected, executing the command ***xpctargetspy*** should display the following:

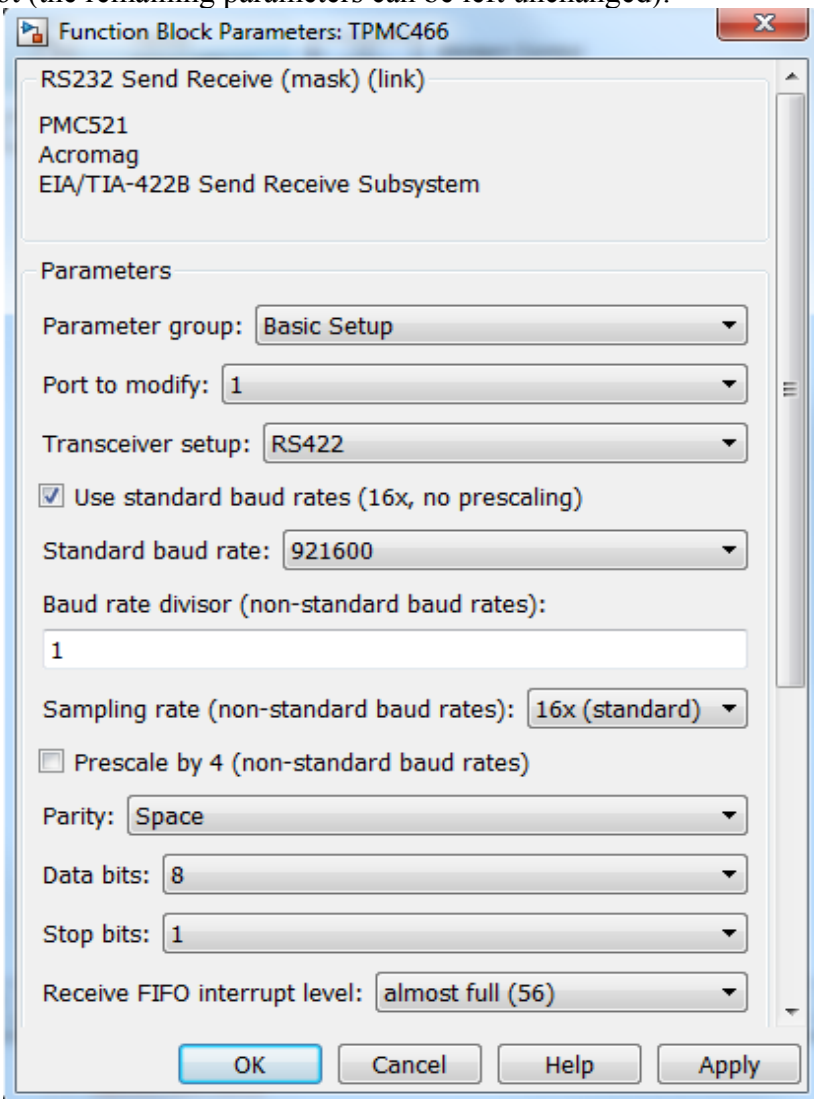


3. Inertial Measurement Unit (IMU)

The inertial measurement unit (IMU) used by the cheetah is the KVH Industries 1750 IMU. The IMU communicates with the Speedgoat via RS-422 serial protocol at 912600 baud rate and 1000 Hz data rate. The Speedgoat is equipped with and IO board (named IO503) which supports RS-422 at the mentioned baud rate. The data rate can be adjusted between 10 Hz and 1000 Hz. Since the Simulink model runs at a sampling rate of 1 ms, then a data rate of 1000 Hz is suitable. To change the data rate on the IMU, please consult the IMU documentation.

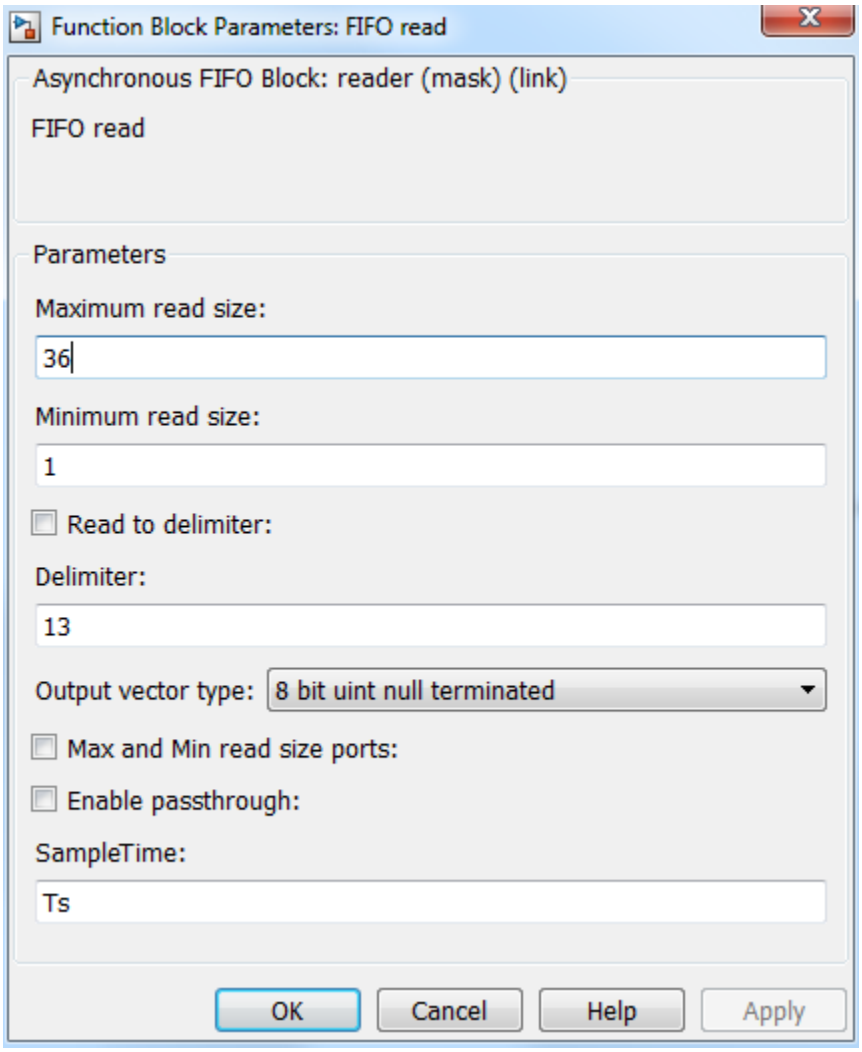
3.1 Setting the Speedgoat to Receive from the IMU

The Speedgoat will communicate with the IMU over the IO503 board. From the Simulink Library Browser, open the xPC Target Speedgoat Driver I/O library then the IO503 library. From there, select the TPMC466 block since the Speedgoat uses this board. Double click the TPMC466 block and from the "Parameter Group" select "Board Setup". Make sure the IRQ number is set to 5 and the PCI slot can be left as -1. Then, select "Basic Setup" and change the settings to match what is shown in the following screen shot (the remaining parameters can be left unchanged):



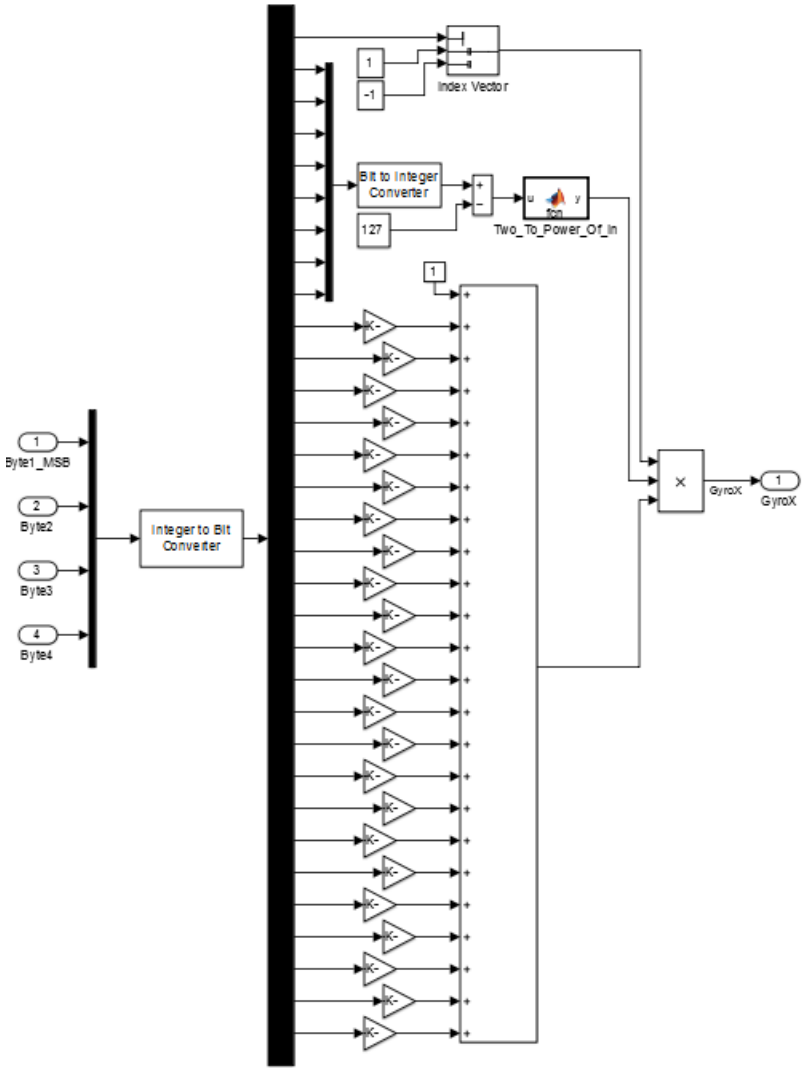
3.2 Receiving Data from the IMU

To receive data from the IMU, open the xPC Target library from the Simulink library browser then insert the "FIFO Read" block from the RS232 library. Connect the output FIFO1 of the TPMC466 block to the input of the FIFO Read block. Double click on the FIFO Read block and change the setting to match the following



3.3 Processing the IMU Data

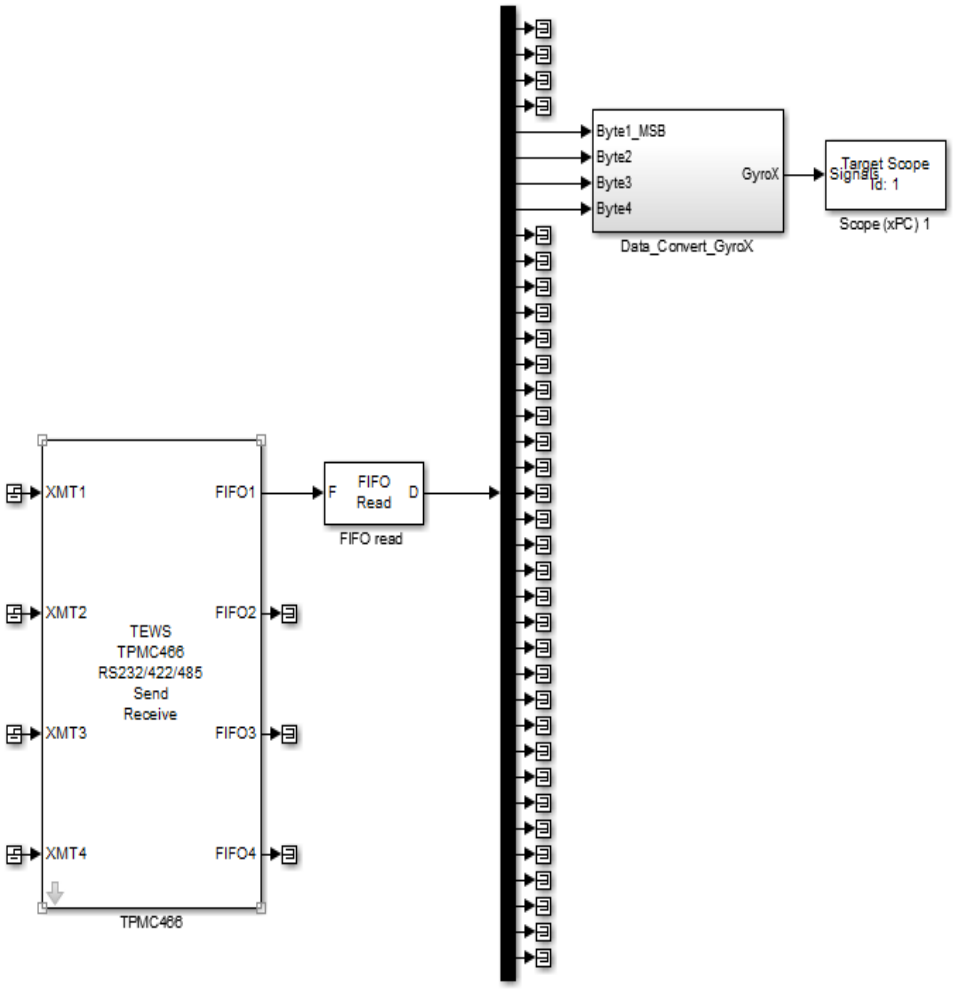
The IMU sends 36 bytes. However, the output of the FIFO Read block will be 37 bytes with the last byte being zero since the data is null terminated. Connect a Demux block with 37 as the output. From the IMU data sheet, it is known that the first 4 bytes are the message header and those are of no interest to us. Suppose the roll (Gyro X) data is desirable which are the next 4 bytes. These four bytes are the input into another subsystem which converts these four bytes into a single precision floating number. This subsystem is shown below



In a single precision floating point number, the first bit (of the 32 bits) is the sign bit. If the bit is zero then sign is positive otherwise it is negative. The next 8 bits are the exponent. They are converted into an integer and subtracted from 127 (which is referred to as the bias). The exponent is then computed are 2^*difference*. To compute the exponent, a user defined MATLAB block was inserted and a function was written since the default power function block in Simulink caused an overflow in the Speedgoat which resulted in an extremely large exponent. This overflow bug doesn't happen often so the reader is advised not to use the Simulink power function block. The code in the screen shot below was used to calculate the exponent. Next, the remaining 23 bits are used to calculate the mantissa. As can be seen above, each bit is multiplied by a gain. The first gain is 1/2^1 and the power keeps increasing all the way to 1/2^23. These gains are all summed up in an addition block and a constant of 1 is added to them. Then, the sign, exponent, and mantissa are multiplied together to get the single precision floating point number.

```
function y = fcn(u)
if u == 0
    y = 1;
    return
end
sign_u = sign(u);
abs_u = sign_u*u;
y=1;
for ii=1:abs_u
    if sign_u==1
        y = y*2;
    else
        y = y/2;
    end
end
```

The final overall Simulink model to read (Gyro X) data from the IMU and display it on a scope should look like the following.

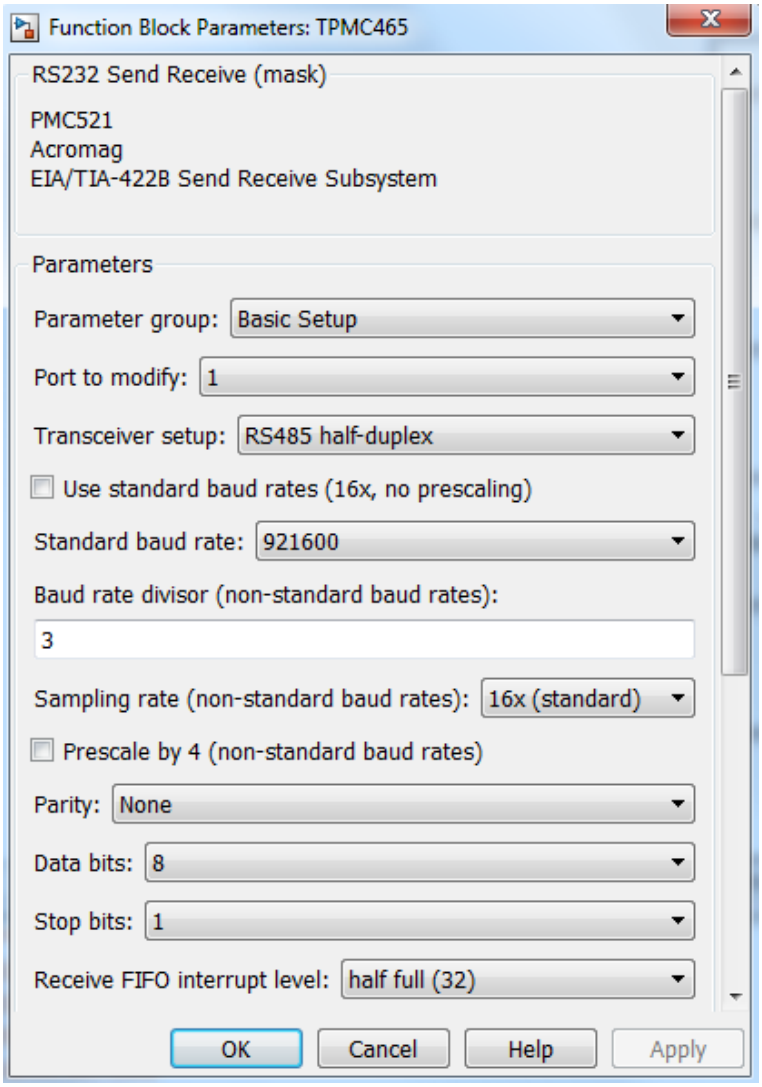


4. Controlling the Dynamixel Motor using Speedgoat

The dynamixel motor uses RS-485 serial protocol to communicate with the Speedgoat at 1000000 baud rate. The Speedgoat has an IO504 board with a custom 48 MHz crystal that allows communication over RS-485 using the non-standard baud rate that the dynamixel uses.

4.1 Setting the Speedgoat to Send to the Dynamixel

The Speedgoat will communicate with the IMU over the IO504 board. From the Simulink Library Browser, open the xPC Target Speedgoat Driver I/O library then the IO504 library. From there, select the TPMC465 block since the Speedgoat uses this board. Double click the TPMC466 block and from the "Parameter Group" select "Board Setup". To use IO504(1) on the Speedgoat set the IRQ number to 9 and PCI slot to [4,5]. To use IO504(2) on the Speedgoat set the IRQ number to 10 and PCI slot to [4,6]. Next, select "Basic Setup" and change the settings to match what is shown in the following screen shot make sure to check the "RS485 auto turnaround" box (not shown below):



Next, from the “Parameter Group” select “FIFO Setup”. Then change the “Transmit FIFO data type” to “count+16 bit uint”. Note, the data being transmitted is still 8 bit unsigned integer. This change is necessary because the default “8 bit uint null terminated” will terminate the transmission of the first null (0x00) is encountered. For instance, if 0x00 was part of the message (such as turning off the LED on the dynamixel) then the remaining of the message (the checksum) will not be transmitted. Notice, the data type needs a “count” block added, this will be further discussed in the next section.

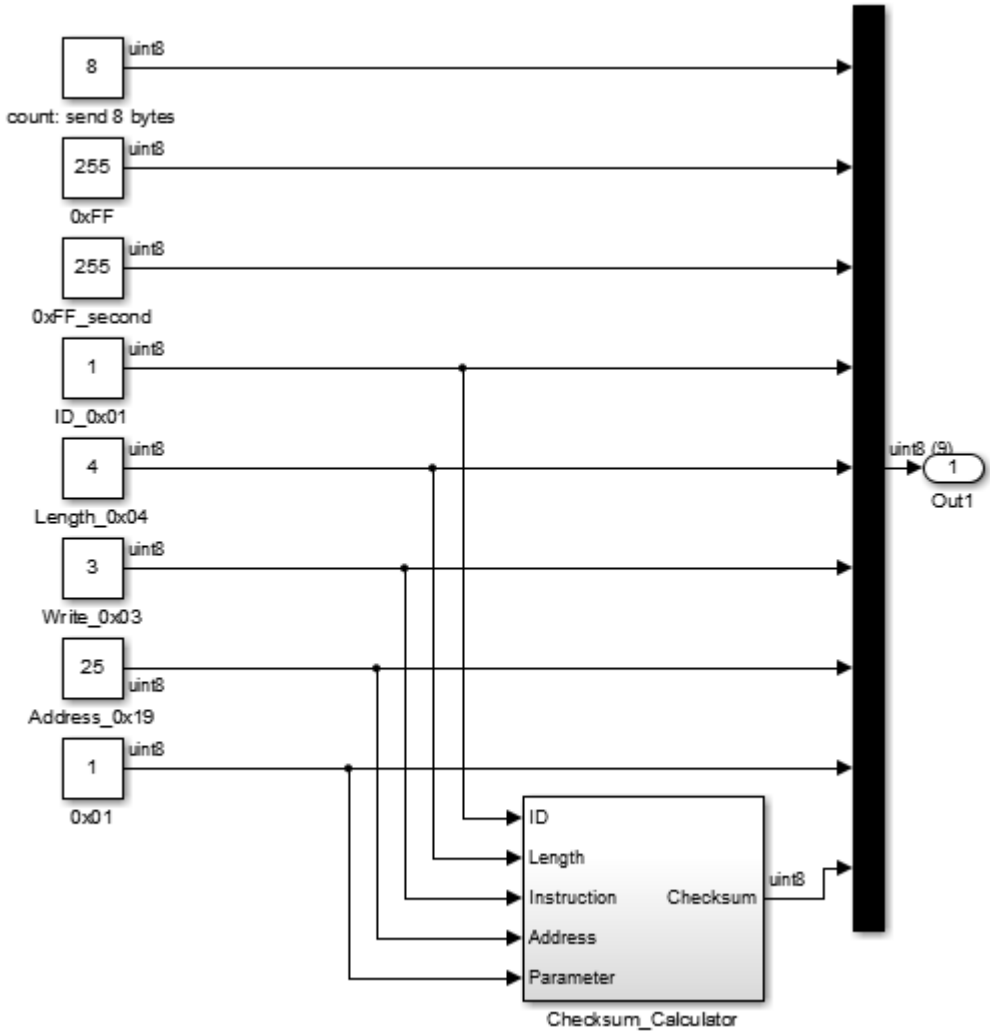
4.2 Sending a Message to the Dynamixel to turn on the LED

To turn on the LED, the following message is sent (in hexadecimal):

FF FF 01 04 03 19 01 D3

Byte(s)	Data (in hexadecimal)	Significance
1 and 2	FF FF	Start of any message sent to dynamixel
3	01	This is the identifier of the dynamixel
4	04	Length of the message since it can vary
5	03	A write command
6	19	The address where the LED on command is sent
7	01	Turn on LED
8	D3	Checksum

To learn more about the identifier, message length calculation, read/write commands, addresses, and calculating the checksum, please refer to the documentation of the dynamixel. Since the message to turn on the LED is 8 bytes in length, then a constant block can be added to the Simulink model with '8' as the constant. The output of all these blocks can then be connected to the input of a Mux block. The output of the Mux block can be connected to the first XMT input of the TPMC465 block to transmit it on channel 1. The Simulink model shown below will create the message and transmit it to the dynamixel. Note, the reader needs to keep in mind that Simulink blocks only work with decimal and do not work with hexadecimal numbers.

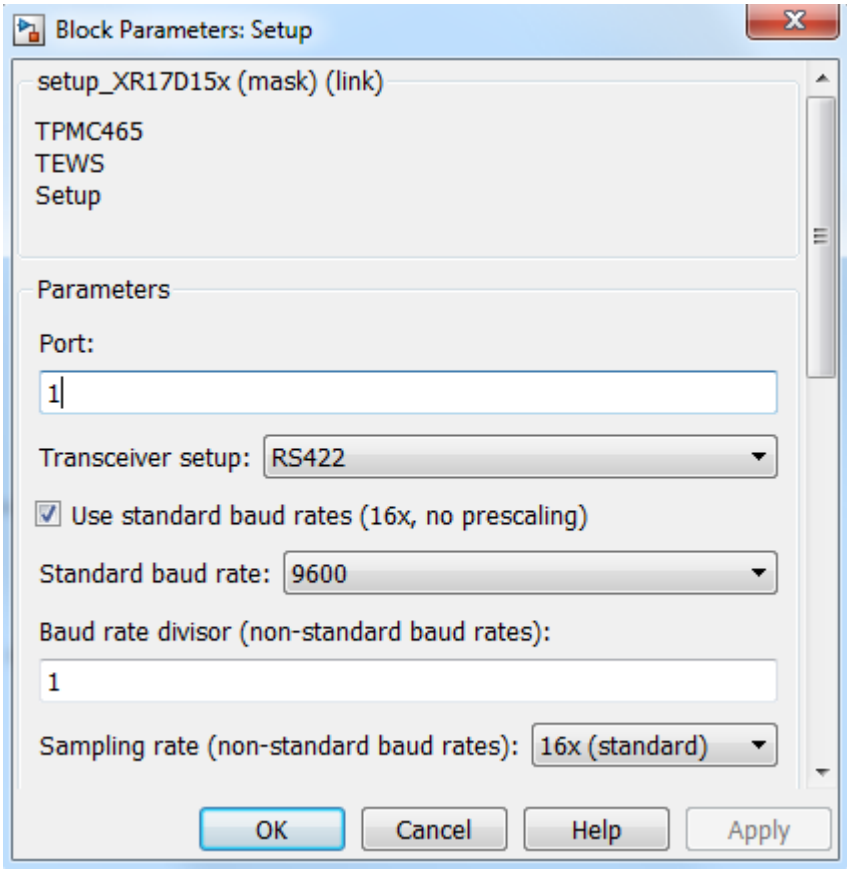


5. Communicating with the Motor Driver

The motor driver has been developed internally at MIT. It communicates with the Speedgoat at 1000000 baud rate. However, at the time of creating this report, the driver used 9600 baud rate. Therefore, this section will demonstrate how to communicate the driver with the Speedgoat using 9600 baud rate. The reader can refer to Section 4 of this report to change the baud rate to 1000000.

5.1 Setting the Speedgoat to Communicate with the Driver

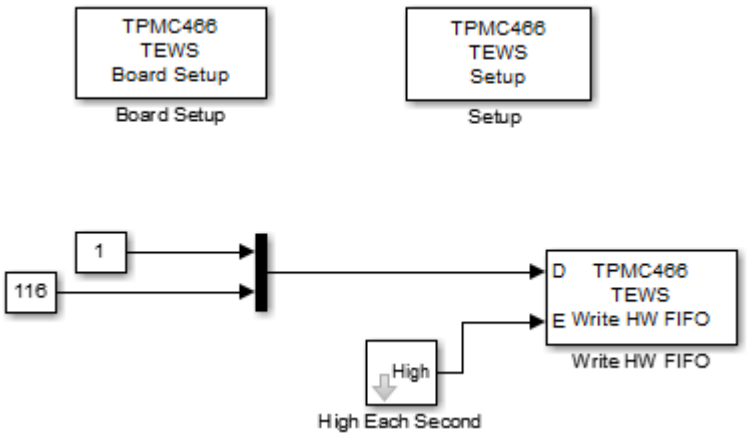
The Speedgoat will communicate with the driver using the IO503 board. Since a baud rate of 9600 will be used, the model needs to be slowed down from 1 kHz to 500 Hz (or Ts = 0.002 s). To change the sample time Ts, please refer to Section 2.1. From the Simulink Library Browser, insert the Board Setup block from the TPMC466 Library under IO503 which is under the xPC Target Speedgoat IO Library. From the Board Setup block, select 5 for the IRQ Line Number. Next, insert a Setup block then open it and change the settings to match the following (the remaining settings can be left alone).



Note: although the driver uses RS485 to communicate with the Speedgoat, we will be using RS422. The Speedgoat uses 4 pins to (2 to transmit and 2 to receive) when using RS422 for each channel. However, if you use RS485 on the Speedgoat, then only 2 pins are active for both send and receive. Additionally, the driver has been developed to receive on one RS485 channel and transmit on another channel. Therefore, to communicate with the driver, the Speedgoat will waste two channels per driver if RS485 is selected but will use only one channel is RS422 is selected.

5.2 Toggling LED2 on the Driver's Board

Insert a Write HW FIFO block from the same library location mentioned in Section 5.1. This block has two inputs. Data is connected to the 'D' input and an enable is connected to the 'E' input. First, to toggle LED2, we send the character 't' in ascii (which is 116 in decimal or 0x74 in hex). However, a 'count' block needs to added and both blocks are connected to the input of a Mux and the output of the Mux to the Write HW FIFO. This is shown in the screen shot below.



We wish to toggle LED2 twice every second. Therefore, we will need the enable input 'E' to go high for one sample period twice each second. For this, we can use a Signal Generator block (from the Sources Library) and have the signal generator output a square wave with a frequency of 4 Hz. The output of generator is connected to the input of a Detect Change block which is found under the Logic and Bit Operations Library. A Data Type Convert block might also be needed. This can be created into a subsystem and the output is connected to the enable 'E' as shown in the screen shot above. Next, connect Pin 3 of the IO503 breakout board to Pin 6 of the Ethernet breakout board. Also, connect Pin 2 from IO503 to Pin 5 for the driver. The pin description can be seen in the appropriate documentation for each the IO503 and the driver.

6. Conclusion

Model-based design is used for faster rapid control prototyping (RCP). The xPC target real-time environment was setup on a Speedgoat computer to facilitate RCP for the second generation MIT Cheetah Robot. A Simulink model was created for communication between the Speedgoat and each of the following: inertial measurement unit (IMU), dynamixel motor, and motor driver.

7. Future Work

The xPC Target real-time environment is now ready to be installed on the second generation MIT Cheetah robot. The motor driver that was bench tested used a baud rate of 9600 but the motor driver on the Cheetah will use 1000000 baud rate. Further work needs to be done to mount each of the IMU and the Speedgoat on the Cheetah. After wards, further development and design of algorithms can be tested using this established environment.

For further questions or inquiries, the author of this document can be reached at zakahmad@gatech.edu