Creating FPGA-based Co-Processors for DSPs Using Model Based Design Techniques

Lab 2

December 2008
Version 3.0

Lab Objectives

Upon completion of this lab, you will …

- Better understand the design exploration process for target hardware through ROI processing, fixed-point numeric computation, and column major data organization
- Understand the c-code generation process and optimization options
- Understand the integration with the TI DSP Design Flow

Understanding the Lab Structure

Each experiment is designed to address one of the above objectives. As such, each section contains the objective, a problem description, and a procedure. The procedure has been broken down into tasks, questions, how-to details, and general tips for future and immediate reference.

Lab Setup

This lab will require the following software and hardware setups.

Software

The software requirements for this lab are:

- WindowsXP
- The MathWorks MATLAB/Simulink R2008a
  - Video and Image Processing Toolbox and Blockset
  - Signal Processing Toolbox and Blockset
  - Real-Time Workshop & RTW - Embedded Coder
  - Embedded IDE Link CC
  - Target Support Package TC6
- Texas Instruments Code Composer Studio

Hardware

The hardware required for this lab is:

- Computer with 1 GB RAM
- Avnet Spartan 3A-DSP FPGA Davinci Development Kit
- Blackhawk USB/JTAG Emulator

Experiment 1: Design Exploration

Objective:
For this experiment we want to model the fixed-point behavior, data management, and data flow of our target hardware.

Problem Description:
We have successfully modeled the behavior of our stabilization algorithm and this provides an executable specification of what we want implemented in the hardware. We now need to perform further exploration and design elaboration to capture our desired implementation and target hardware. These design considerations include fixed-point computation, data organization/management, hardware accelerators and intrinsic utilization, peripheral device utilization, and ROI based processing.

Procedure:
1. Region of Interest (ROI) Processing
    - Limit the search region
2. Fixed-Point Conversion
    - Identifying blocks to be converted
    - Data Type Override
    - Data Logging
    - Fixed-Point Scaling
    - Improving the simulation accuracy

Region of Interest (ROI) Processing

Tasks:

- Limit the search region

Details:

Limit the search region

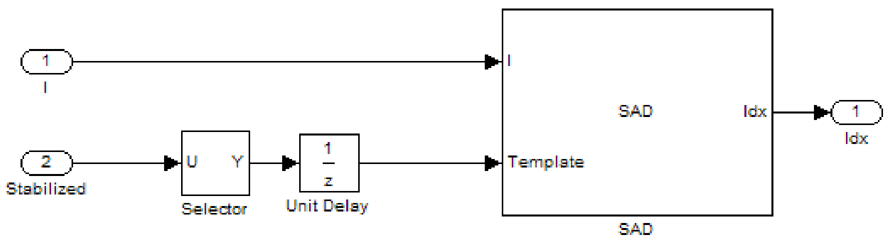
By limiting the size and establishing a dynamic search region where the position is determined by the last known target location, we can dramatically reduce the number of computations required to find the target. This can also benefit our algorithm further, because with a smaller search region there will be less cache misses, and later we may choose to utilize DMA to move the data block.

Action Open the SAD_Stabilize.mdl model in the lab 2 solutions directory.

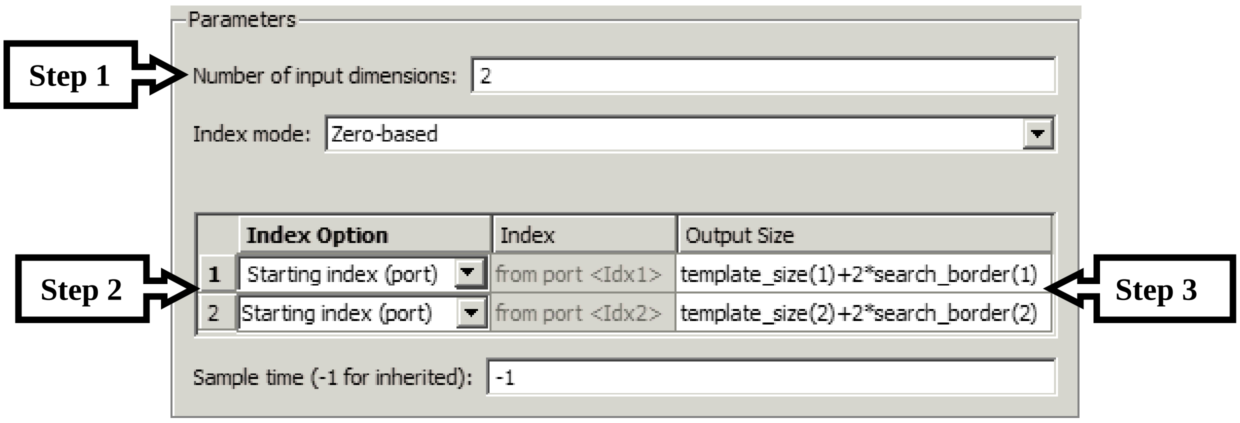
To limit the search region, we first need to crop a smaller search region around the template and provide this as the image to the SAD algorithm.

Action Open the Estimate Motion Subsystem and add the following blocks as shown:

Block	Library	Quantity
Data Type Conversion	Simulink > Signal Attributes	1
Selector	Simulink > Signal Routing	1
Demux	Simulink > Signal Routing	1
In1	Simulink > Sources	1



The selector block will select a region defined by the following parameters:



Parameters

Number of input dimensions: 2

Index mode: Zero-based

	Index Option	Index	Output Size
1	Starting index (port)	from port <Idx1>	template_size(1)+2*search_border(1)
2	Starting index (port)	from port <Idx2>	template_size(2)+2*search_border(2)

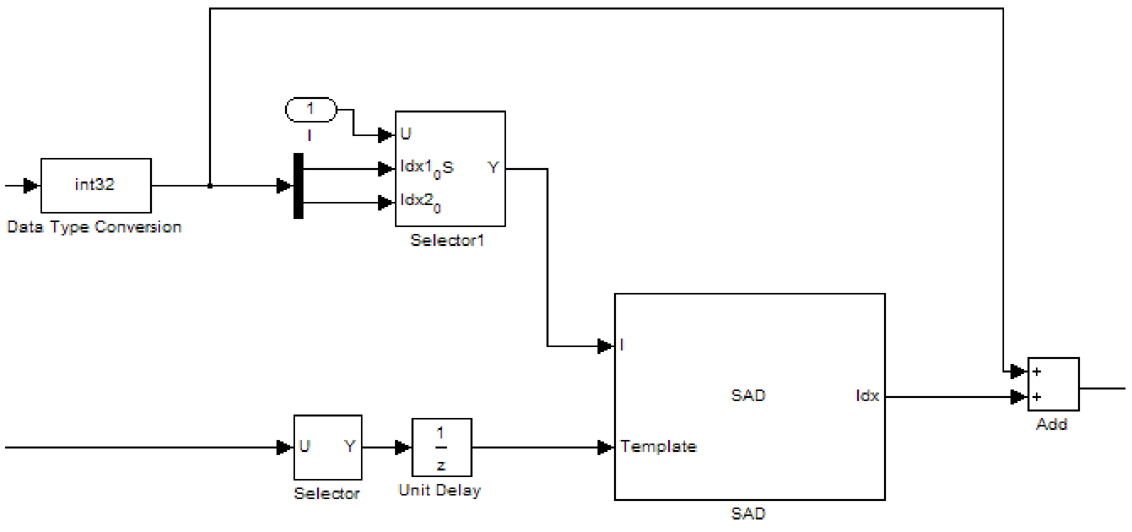
Sample time (-1 for inherited): -1

The `search_border` parameter is defined in the MATLAB workspace and was created through a model callback at the time we opened the model. The value is [25 25].

Note: *The SAD block has an ROI processing capability that is more efficient for modeling and implementation. The ROI is defined by the upper left corner and the size. We will utilize this capability later.*

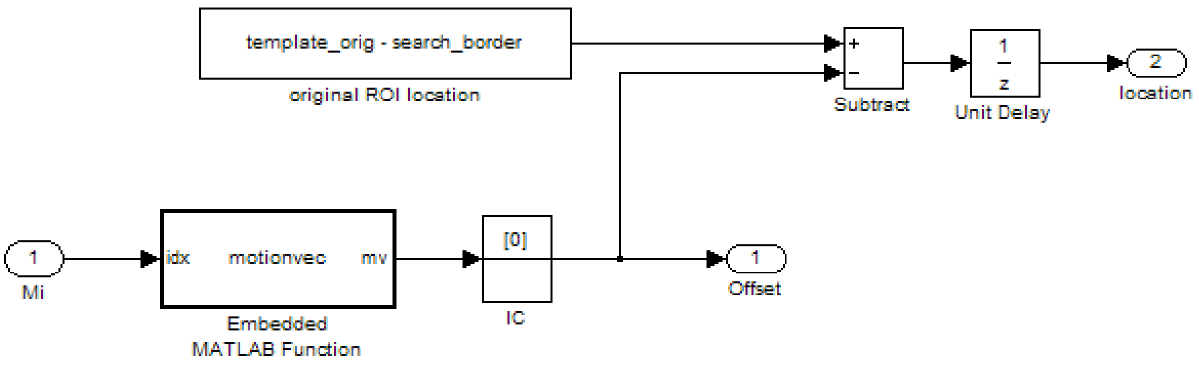
Action Now that the input image to the SAD block is actually an ROI from the larger image, we need to adjust the index to account for the search region. This can be done with a simple add block.

Step 1

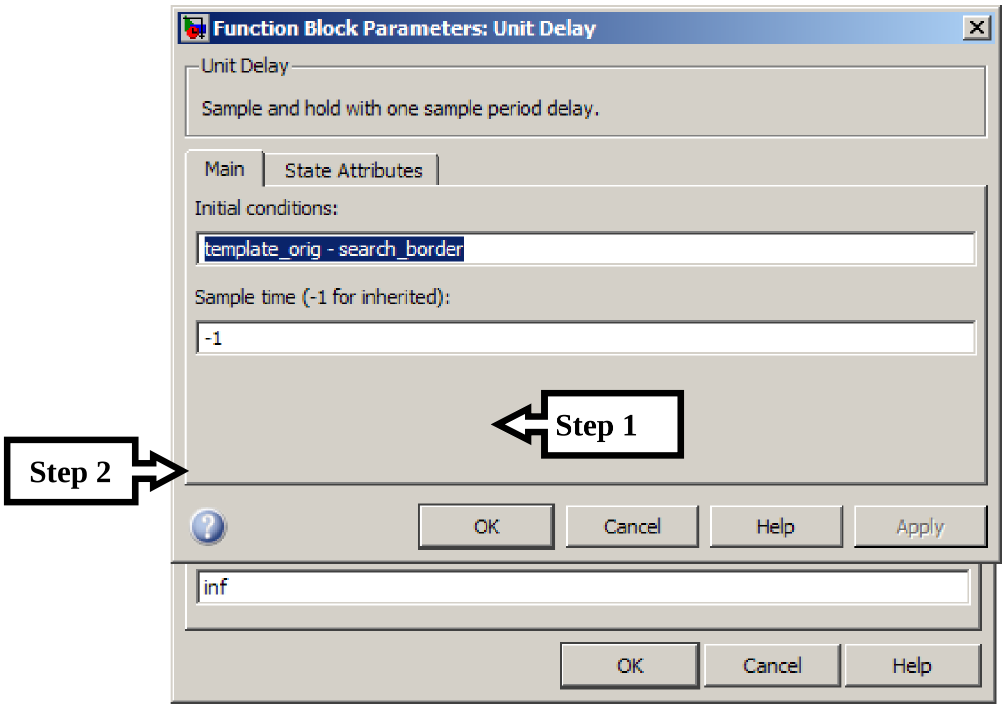


Action The Update subsystem needs to calculate the position of the upper left corner of the search region. We need the following blocks:

Block	Library	Quantity
Constant	Simulink > Commonly Used Blocks	1
Unit Delay	Simulink > Discrete	1
Subtract	Simulink > Math Operations	1
Out1	Simulink > Sinks	1



The unit delay block should be set as follows:



Function Block Parameters: Unit Delay

Unit Delay

Sample and hold with one sample period delay.

Main | State Attributes

Initial conditions:

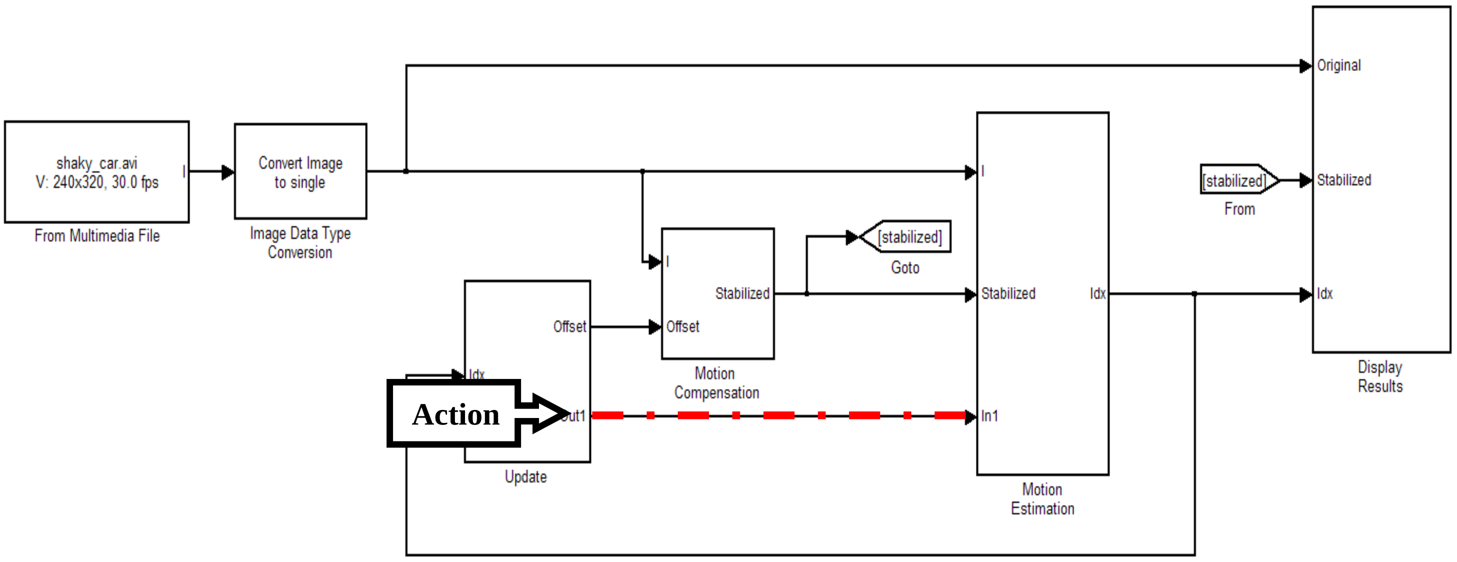
template_orig - search_border

Sample time (-1 for inherited):

-1

inf

Finally, connect the top level model as shown below and simulate the results:



Questions to guide exploration:

1. What is the selector block output size?

2. How is the performance of the ROI processing as compared to the original? Is it faster? Are the results the same?

3. Let the simulation run for a while. Are there any effects of performing ROI processing with a dynamic search region?

Fixed-Point Conversion

Tasks:

- Identifying blocks to be converted
- Data Type Override
- Data Logging
- Fixed-Point Scaling

Details:

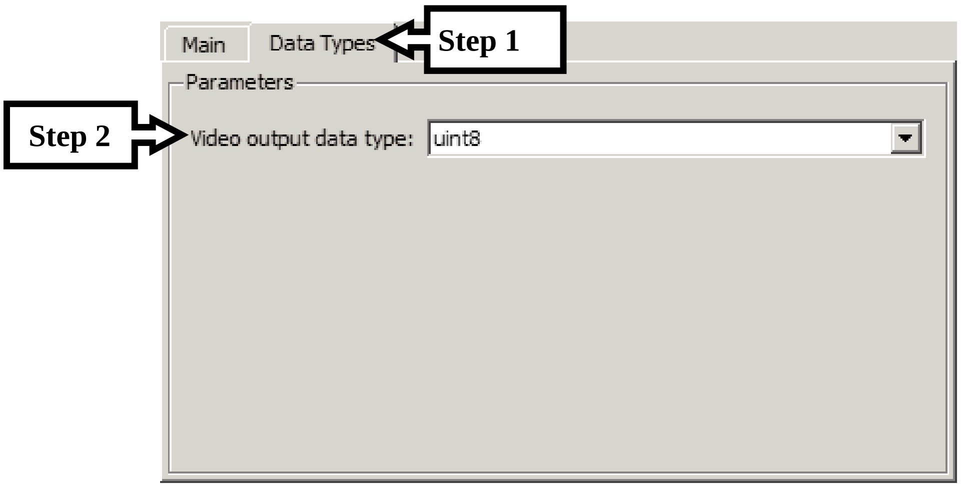
Identifying blocks to be converted

There are multiple approaches for fixed–point conversion, but as a practice, it is best to insert data type conversion blocks around each module that is data type dependant and then work through conversion one module at a time. This method is robust and it allows us to develop unit tests for each module.

Action Open the SAD_Stabilize_ROI.mdl model.

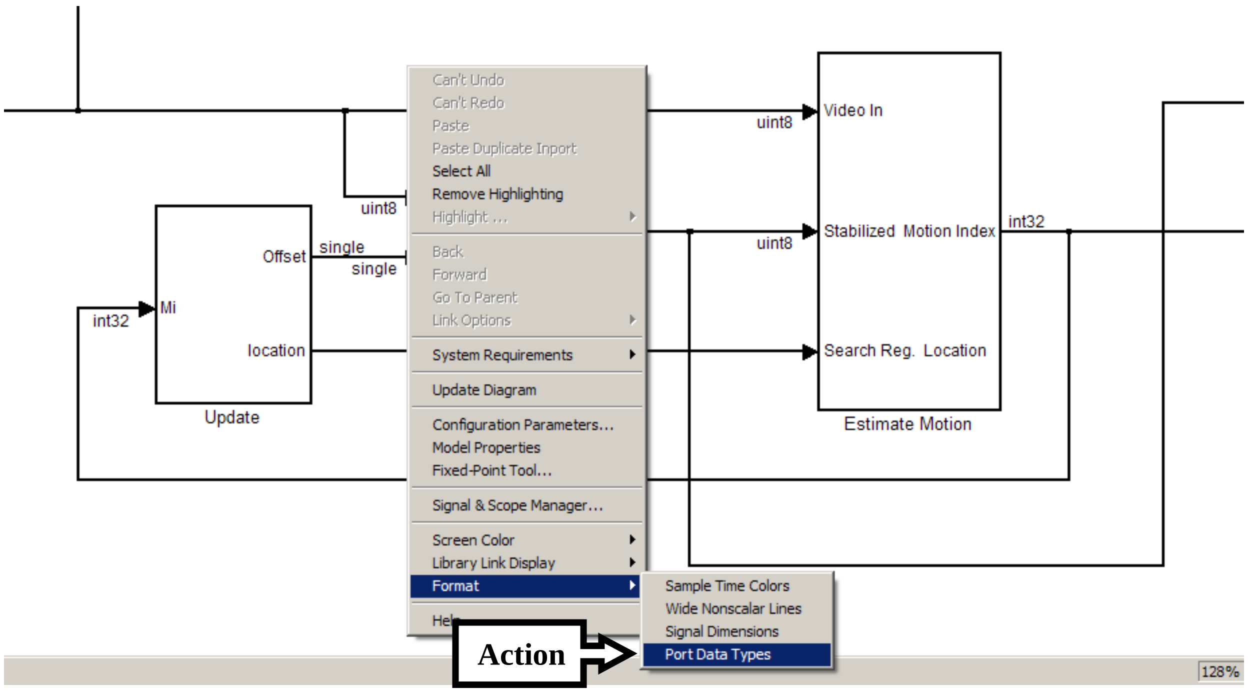
To see the modules that are data type dependant, you can allow the model to update with back propagation. If there are errors at certain blocks, then it may be a good idea to insert a data type conversion. For our algorithm, let us first change the data type of the input source to let the data type propagation update.

Action Change the From Multimedia File block's data type to uint8

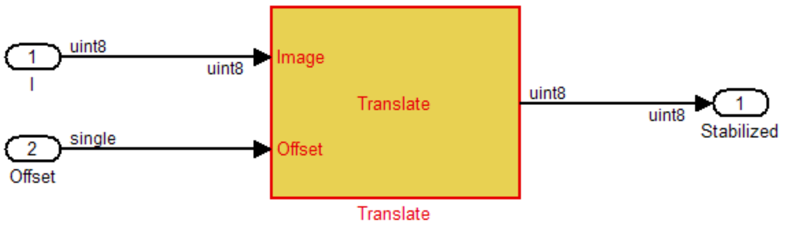


Video output data type: uint8

Action Next, we can visualize the signal data type propagation for each block by the following setting:



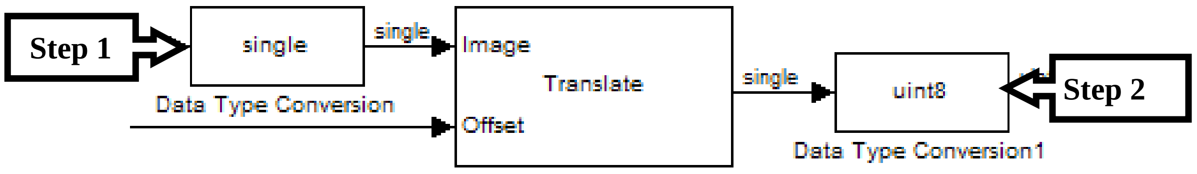
Update the diagram and we should see that the Offset data type for the Translate block is single, but the input image is not.



Note: *The translate block requires all of its inputs to be integer, or floating point, not a mix of both.*

Data Type Override

Action Inserting data type conversions before and after the block will eliminate the errors.



We can also add Data Type Conversions for the other modules, which will enable us to analyze each module independent of the surroundings, but it is not necessary for this particular model.

Note: *The fixed-point tool has a method for data type override that would also work for this process.*

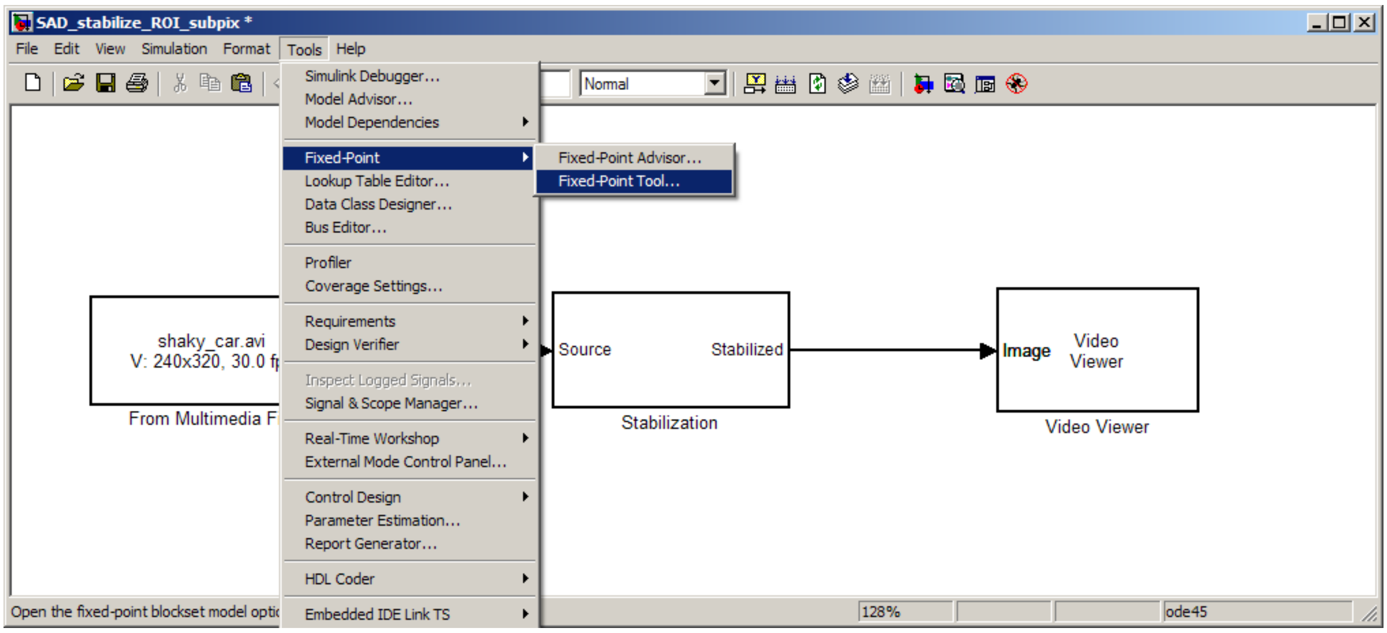
Questions to guide exploration:

1. The simulation errors out after a few frames, what is the cause?

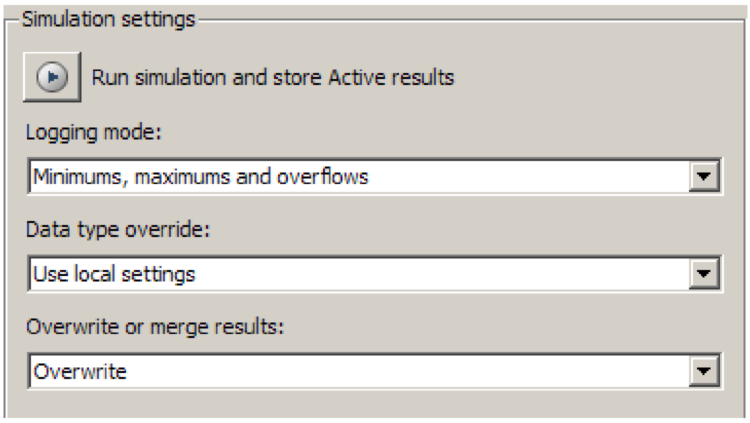
2. Note that the display is white boxes, what causes this and how can we correct it?

Data Logging

Action Open the fixed-point tool (**Tools** > **Fixed-Point** > **Fixed-Point Tool…**):



Action Set the logging Mode to Minimums, Maximums, and overflows, then simulate the model.


Simulation settings

Run simulation and store Active results

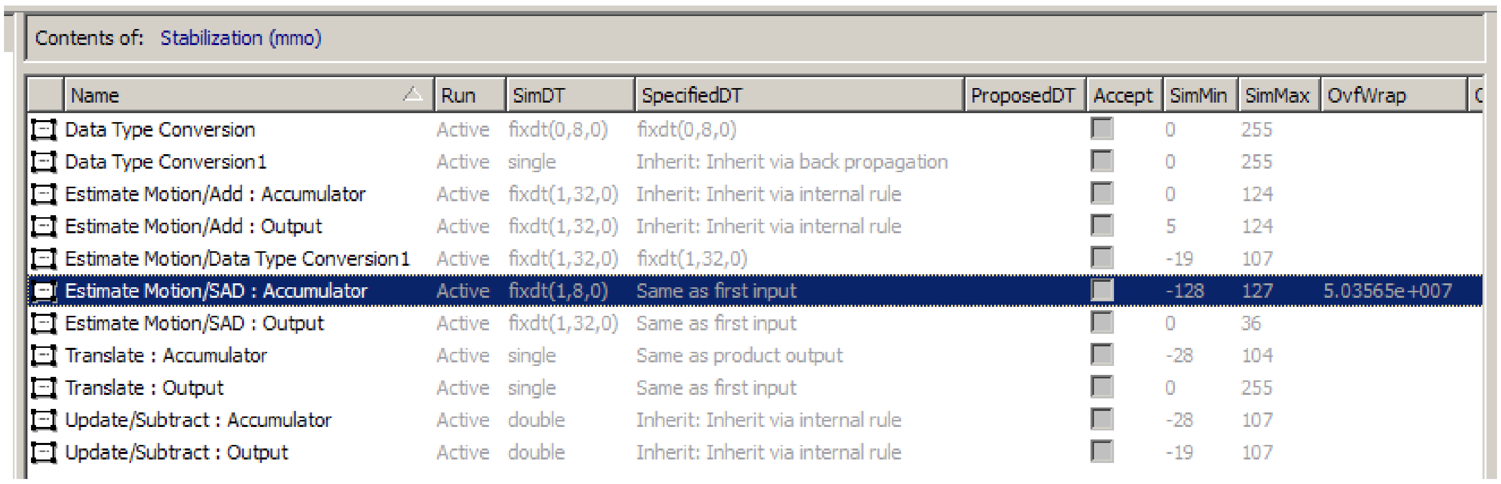
Logging mode: Minimums, maximums and overflows

Data type override: Use local settings

Overwrite or merge results: Overwrite

Fixed-Point Scaling

The Accumulator for the SAD block is overflowing, so we need to adjust the fixed-point settings to accommodate a larger range.



Contents of: Stabilization (mmo)

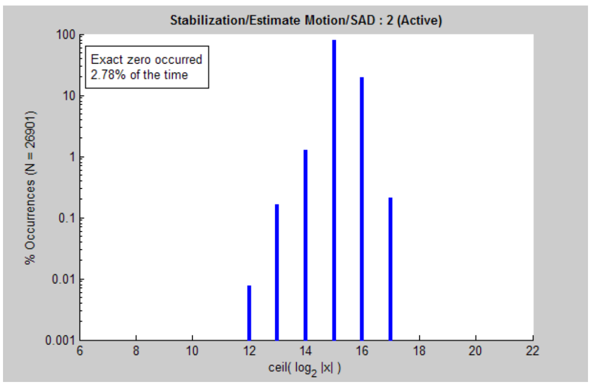
Name	Run	SimDT	SpecifiedDT	ProposedDT	Accept	SimMin	SimMax	OvfWrap
Data Type Conversion	Active	fixdt(0,8,0)	fixdt(0,8,0)		☐	0	255	
Data Type Conversion1	Active	single	Inherit: Inherit via back propagation		☐	0	255	
Estimate Motion/Add : Accumulator	Active	fixdt(1,32,0)	Inherit: Inherit via internal rule		☐	0	124	
Estimate Motion/Add : Output	Active	fixdt(1,32,0)	Inherit: Inherit via internal rule		☐	5	124	
Estimate Motion/Data Type Conversion1	Active	fixdt(1,32,0)	fixdt(1,32,0)		☐	-19	107	
Estimate Motion/SAD : Accumulator	Active	fixdt(1,8,0)	Same as first input		☐	-128	127	5.03565e+007
Estimate Motion/SAD : Output	Active	fixdt(1,32,0)	Same as first input		☐	0	36	
Translate : Accumulator	Active	single	Same as product output		☐	-28	104	
Translate : Output	Active	single	Same as first input		☐	0	255	
Update/Subtract : Accumulator	Active	double	Inherit: Inherit via internal rule		☐	-28	107	
Update/Subtract : Output	Active	double	Inherit: Inherit via internal rule		☐	-19	107	

Figure 11

We can override the data types for the input of the SAD block to determine the range, or we can estimate it through a worst case scenario. In the worst case, the accumulator maximum is the maximum value of the data type times the number of elements. For our case, this is 255*18*22 or a little over 100k. This requires 17 bits of precision.

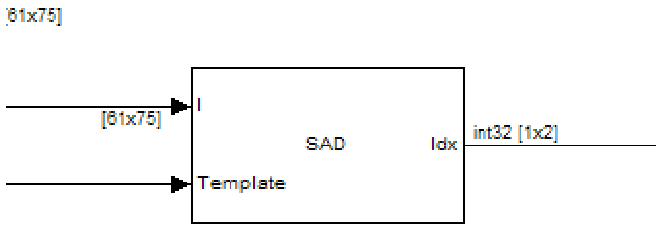
To override the data type and log the range, we can create a subsystem for the stabilization algorithm and limit the override to just the algorithm. When we do this, we can see that the maximum value is approximately 96k. This accounts for the first frame, which is near the worst case.

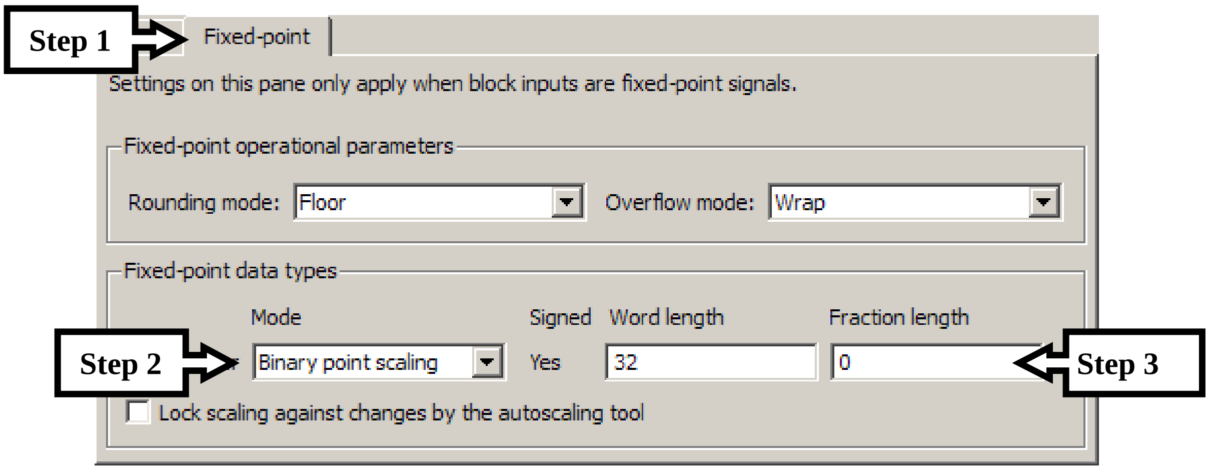
We can further narrow the result by outputting the SAD value matrix and logging the data. This will allow us to create a histogram plot:



This plot shows us that less than 1% of the data requires more than 16 bits. Therefore we have the choice to accommodate the full range and choose 32 bits for the accumulator value, or we can use 16 bits (with saturation) and have a small percentage of data overflow.

Action Set the accumulator value of the SAD block to accommodate the full range.





Settings on this pane only apply when block inputs are fixed-point signals.

Fixed-point operational parameters

Rounding mode: Floor   Overflow mode: Wrap

Fixed-point data types

Mode	Signed	Word length	Fraction length
Binary point scaling	Yes	32	0

☐ Lock scaling against changes by the autoscaling tool

Simulate the model and note the overflows.

Note: *For a faster simulation, try accelerator mode.*

There is no need to complete the remainder of the fixed-point conversion, this has already been completed and the new model is located in the solution directory.

```
>> SAD_stabilize_ROI_fixpt.mdl
```

Improving the simulation accuracy

The fixed-point precision and the dynamic search region are both major sources of error propagation in the algorithm. The effects are accumulated and over time we may start to notice drift in the video frames. There are a variety of ways to mitigate this type of error, one possibility is to perform sub-pixel motion estimation and compensation.

The SAD computation by itself is not capable of sub-pixel accuracy, so one way we can estimate sub-pixel motion is to fit a surface to the SAD value neighborhood surrounding the minimum index. The location of the minimum surface peak can be calculated with sub-pixel accuracy and this location can be used as an offset for our motion vector estimate.

Sub-pixel accurate motion also allows us to use more advanced interpolation methods for the motion compensation and this will produce pixel values that are on a continuous range as opposed to quantized 8 bit integers. As such, the fixed-point settings will all need to be adjusted and we may want to disable the sub-pixel estimate for comparison. This more advanced model can be found in the solution directory and is the model we will use for the remainder of the lab.

Action Open SAD_stabilize_final_fixpt.mdl

Note: *This model uses the ROI input of the SAD block, performs sub-pixel motion estimation through a surface fit, has been converted entirely for fixed-point, and incorporates a bound check for the search region so that the model does not index out of bounds.*

Action Explore the model and note the simulation results.

This concludes Experiment 1

Experiment 2: C-Code Generation

Objective:
For this experiment, we want to explore the c-code generation process and the different optimization options for targeting the DM6437 processor. You will use Embedded IDE Link CC with Real-Time Workshop® to generate complete projects, including application source code and linker command files, from Simulink models.

Problem Description:
This experiment suggests a development workflow for our fixed-point video application to be deployed on an embedded processor. The workflow goes through generating verified, profiled and optimized code for the selected processor. The tasks are arranged in the suggested order, but are presented in an independent manner that allows you to skip sections as you wish.

Note: *Before continuing, verify that you have configured at least one board in Code Composer Studio™ (CCS). If not, please go to Code Composer Studio Setup and select a board based on a supported TI processor.*

Procedure:
Generate and Profile Code

- Device Driver Integration
- Asynchronous Scheduling Implementation
- Generate & Profile Code
- Optimize Code Using Compiler Options & Hardware Optimized Blocks

Generate and Profile Code

Tasks:

- Device Driver Integration
- Asynchronous Scheduling Implementation
- Generate & Profile Code
- Optimize Code Using Compiler Options & Hardware Optimized Blocks

Questions:

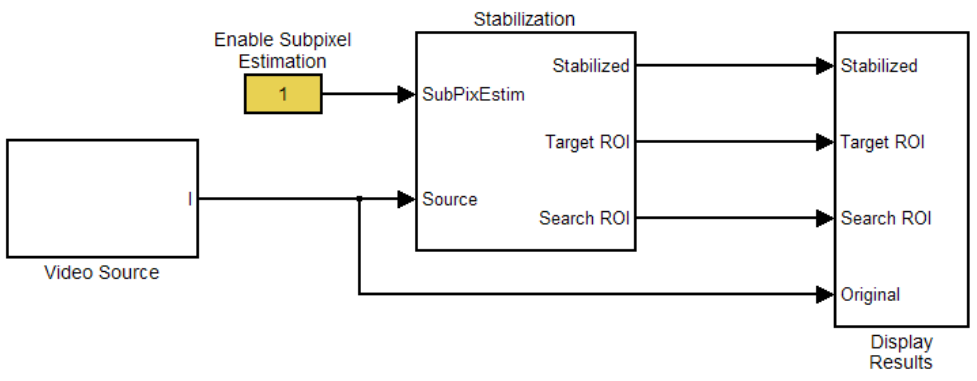
1. What size image is required for the LCD block?
2. What are the names of the signals from the Video Capture block and to the LCD block? How are these resolved in the generated code?
3. What percentage of time and resources does the SAD block require in the stabilization algorithm?
4. What is the difference between average time and maximum time spent? Which number should we utilize for benchmarking purposes?
5. What is the performance improvement by using the –o2 and TFL options in the generated code?
6. What is the optimal stack size?

Details:

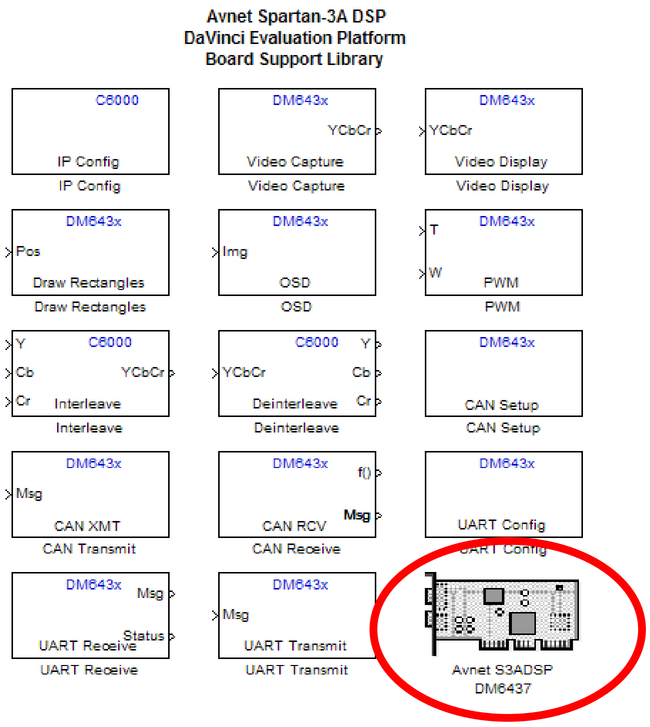
Device Driver Integration

In order to capture and display the video, you will integrate "Video Capture" and "Video Display" blocks with the algorithm.

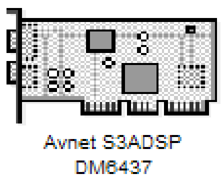
Action Open the SAD_stabilize_final_rowmajor_fixpt_codegen.mdl model.



Action Open the avnet_s3adsp_dm6437 library and locate the Target Preferences block.

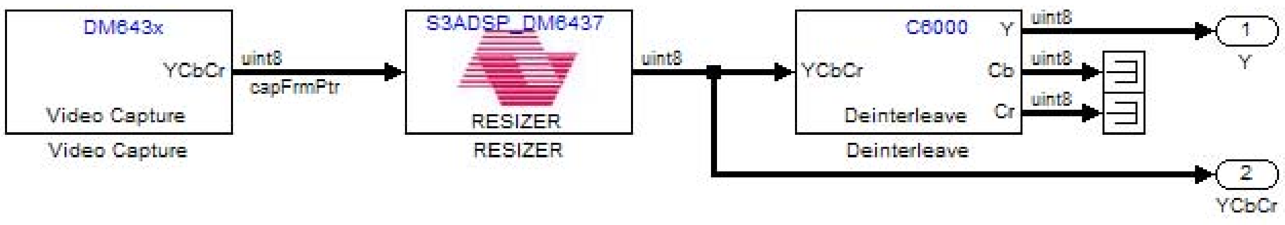


Action Copy the Avnet S3ADSP DM6437 Target Preferences block into the model and select **Yes** in the dialog.

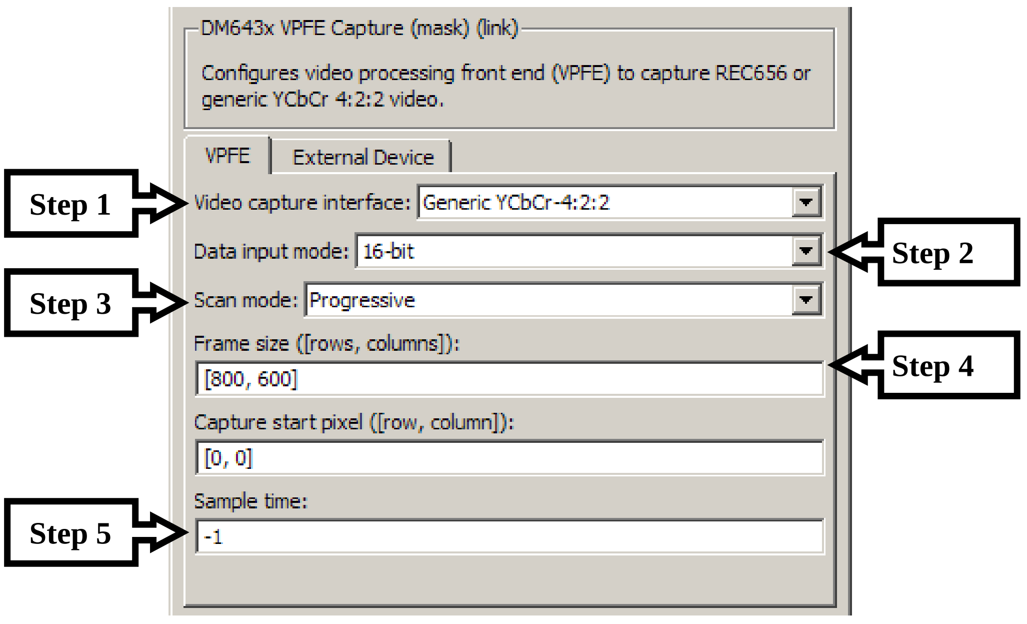


Action Insert Video Capture, Resizer, and Deinterleave blocks inside the Video Source subsystem.

Note: *The Resizer block is located in the avnet_s3adsp_dm6437_dsplib library.*



Action Set the video capture properties to Generic YCbCr-4:2:2 in order to capture video over VGA. The VGA input size should be set to 800x600.



DM643x VPFE Capture (mask) (link)

Configures video processing front end (VPFE) to capture REC656 or generic YCbCr 4:2:2 video.

Video capture interface: Generic YCbCr-4:2:2

Data input mode: 16-bit

Scan mode: Progressive

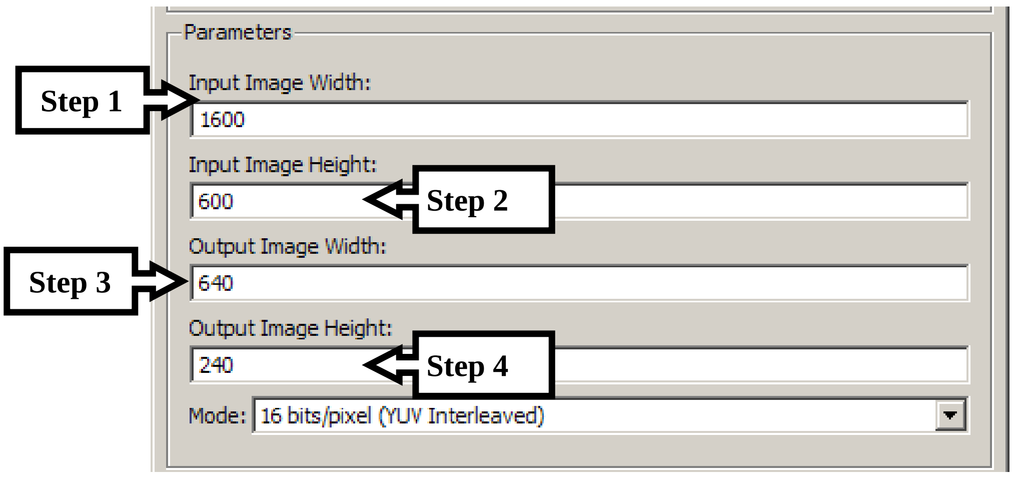
Frame size ([rows, columns]): [800, 600]

Capture start pixel ([row, column]): [0, 0]

Sample time: -1

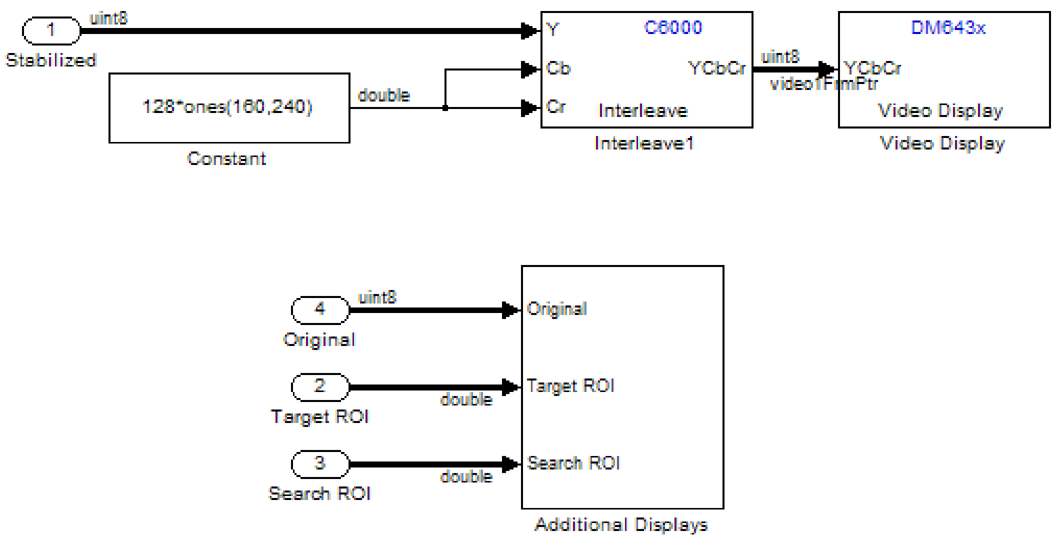
Action The Resizer block should be set to downsize the input image so that the size matches the shaky_car image size (320x240).

Note: *The input image is interleaved and therefore contains 2 times as many columns as the desired size.*

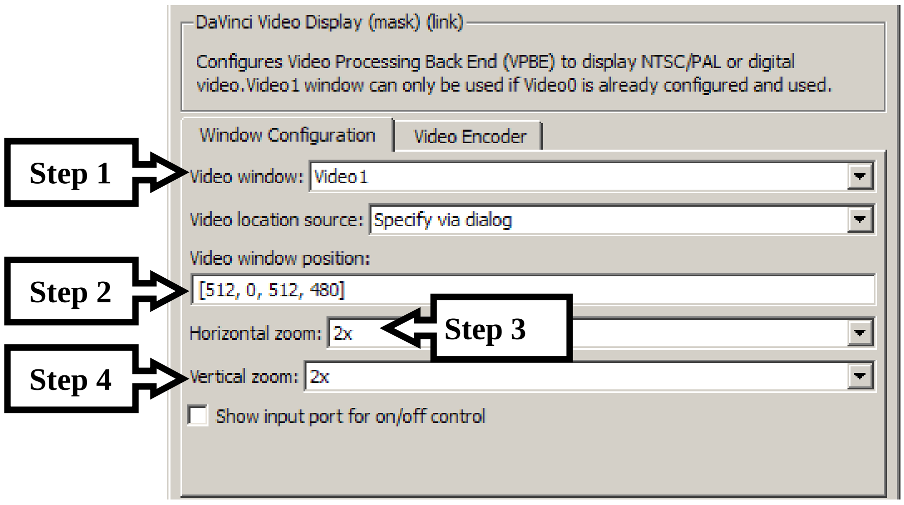


Input Image Width: 1600
Input Image Height: 600
Output Image Width: 640
Output Image Height: 240
Mode: 16 bits/pixel (YUV Interleaved)

Action Insert the Video Display block and the Interleave block inside the Display subsystem.



Set the Video Display block options as follows.



DaVinci Video Display (mask) (link)

Configures Video Processing Back End (VPBE) to display NTSC/PAL or digital video.Video1 window can only be used if Video0 is already configured and used.

Video window: Video1

Video location source: Specify via dialog

Video window position: [512, 0, 512, 480]

Horizontal zoom: 2x

Vertical zoom: 2x

☐ Show input port for on/off control

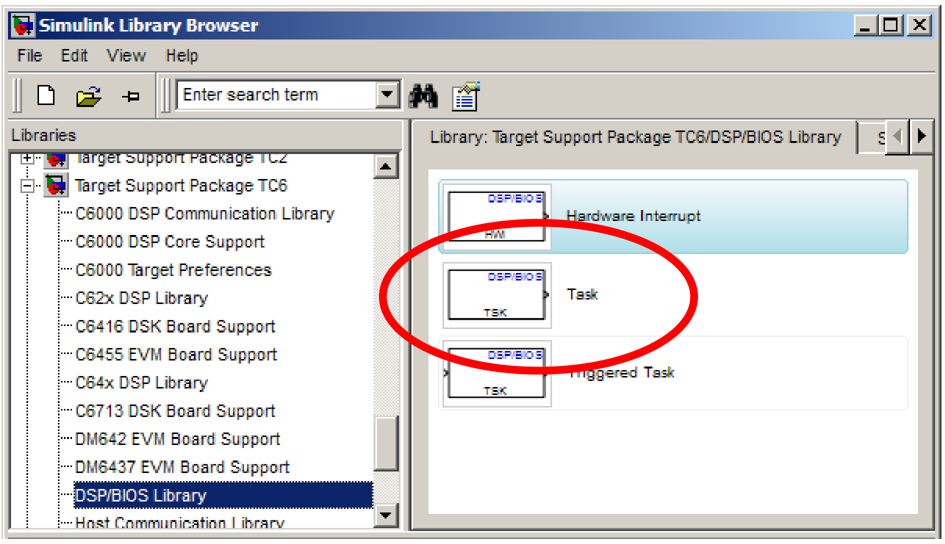
Note: *Inside the Additional Displays subsystem there is a Video Display block that sets up the LCD. The subsystem also contains a Draw Rectangle Block and several OSD blocks in order to output the various calculations from the algorithm.*

Action Generate the code for the model by pressing Ctrl+B. After the program builds, verify that the algorithm works.

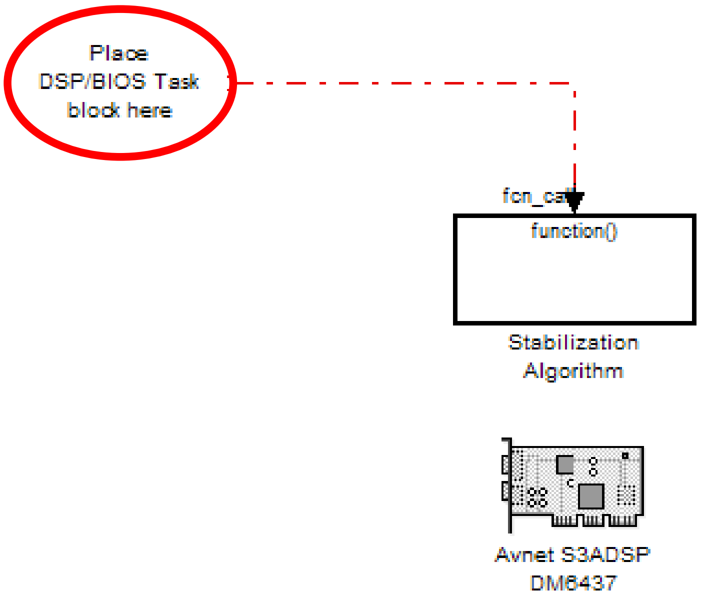
Asynchronous Scheduling Implementation

Action Open the SAD_stabilize_final_rowmajor_fixpt_async.mdl model.

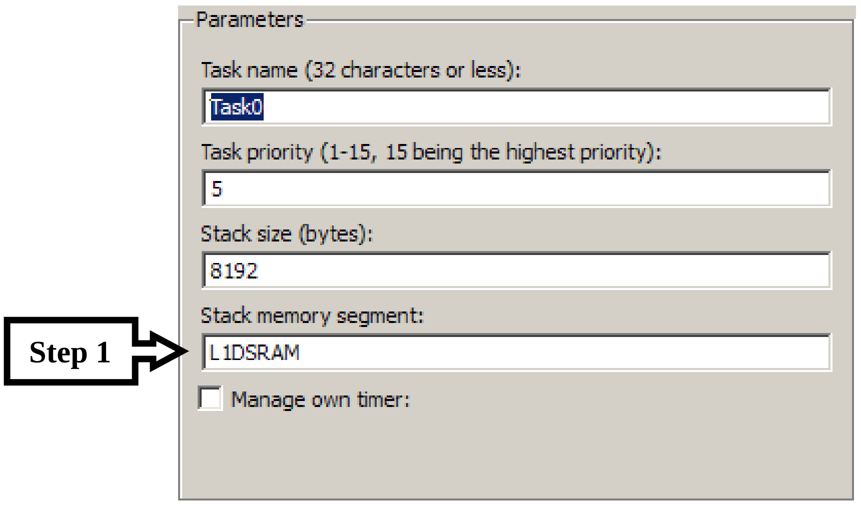
Action Open the DSP/BIOS Support Library and copy the "Task" block to the template model in the location where "Place Task block here" is written.



Action Connect the signal lines to the corresponding output ports of the "Task" block.



Action Set "Stack memory segment" parameter of the Task block. The value should be L1DSRAM.



Parameters

Task name (32 characters or less):
Task0

Task priority (1-15, 15 being the highest priority):
5

Stack size (bytes):
8192

Stack memory segment:
L1DSRAM

☐ Manage own timer:

Action Generate the code for the model by pressing Ctrl+B. After the program builds, verify that the algorithm works.

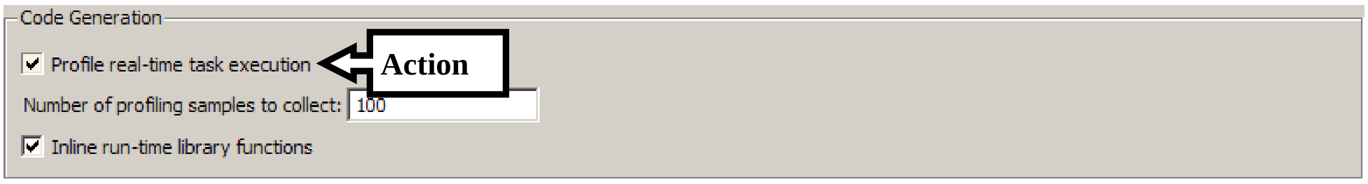
Generate & Profile Code

After verifying the processor implementation results, the next step is to profile the generated code to look for run-time performance improvements.

Note: *Real-Time Task Execution Profiler uses STS objects within Code Composer Studio.*

Action First, configure the model to perform Real-Time Task Execution Profiling by opening **Simulation** > **Configuration Parameters**.

Under **Real-Time Workshop** > **Embedded IDE Link CC**, check "Profile real-time task execution" option.


Code Generation

[✓] Profile real-time task execution

Number of profiling samples to collect: 100

[✓] Inline run-time library functions

Action Click OK to save the configuration and close the dialog box.

Action Generate, build, load, and execute code on the processor. Let the application run for a few seconds, and then halt the processor by clicking the 'Halt Processor' icon in Code Composer Studio.

Action View the profiling results by typing the following command at the MATLAB command prompt:

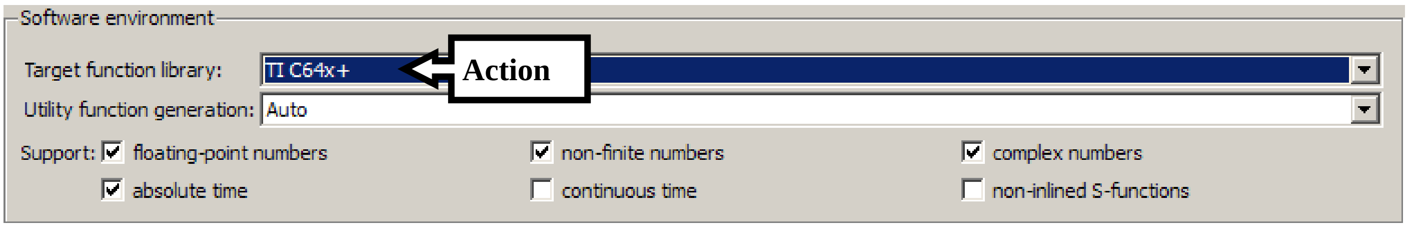
```
>> profile(CCS_Obj, 'report')
```

Optimize Code Using Compiler Options & Hardware Optimized Blocks

The code generation performance can be optimized by enabling the Target Function Library (TFL). This will replace basic fixed-point mathematics with processor intrinsic in the generated code. To enable the TFL, open the **Simulation** > **Configuration Parameters**.

Action

Under **Real-Time Workshop** > **Interface**, select the TI 64x+ Target Function Library option.


Software environment

Target function library: TI C64x+

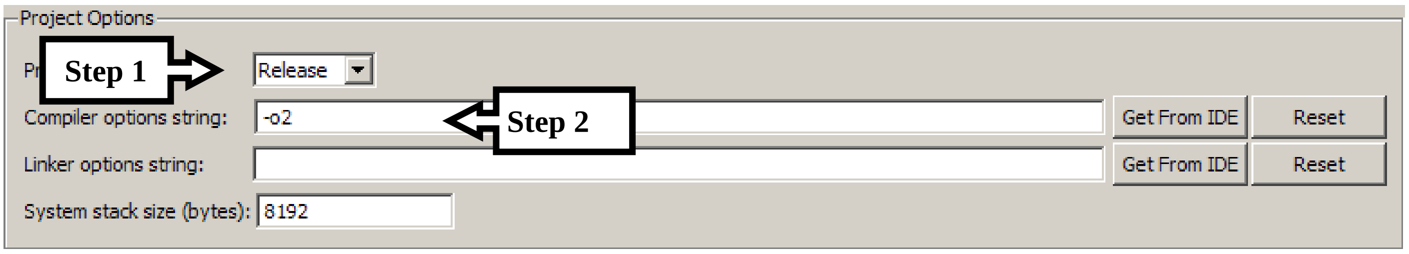
Utility function generation: Auto

Support: [✓] floating-point numbers [✓] non-finite numbers [✓] complex numbers

[✓] absolute time [ ] continuous time [ ] non-inlined S-functions

Action

Next, we can utilize compiler optimizations for Code Composer Studio by updating the Compiler options string under **Real-Time Workshop** > **Embedded IDE Link CC**. For our purposes, the –o2 option is a good level of optimization.


Project Options

Pr… : Release

Compiler options string: -o2

Linker options string:

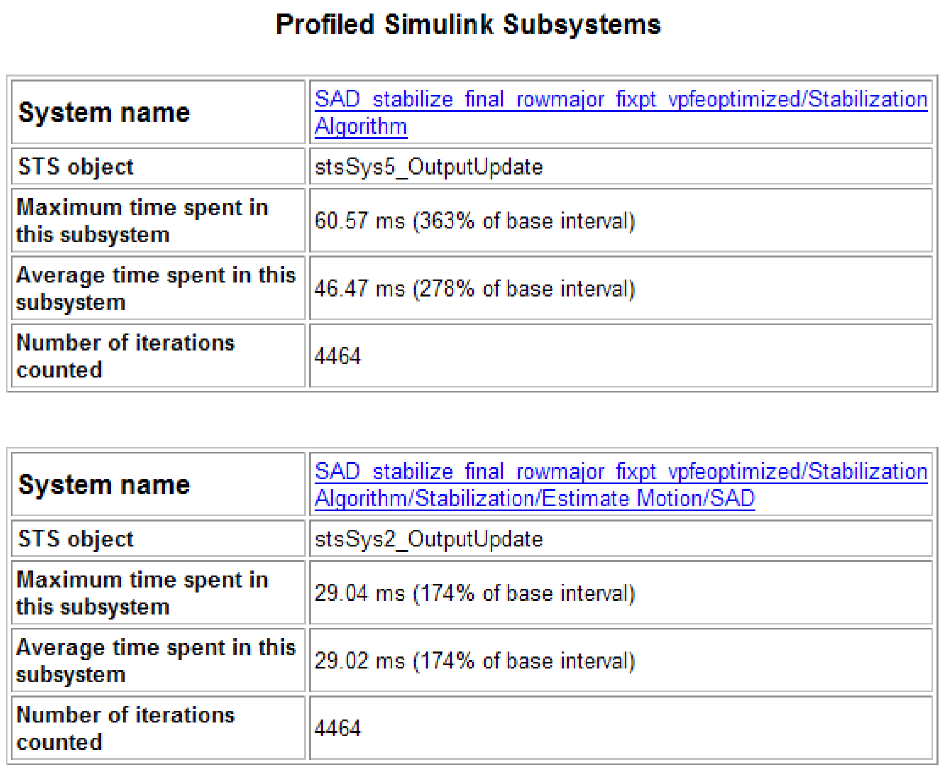
System stack size (bytes): 8192

To optimize the model with hardware blocks, we have multiple possibilities. First is to use the block processing utility with DMA to perform patch processing and data movement for our various components. This solution would require a complete restructuring and remodel to implement properly and we may not gain a lot of MIPS because our ROI is already small; however we can easily implement this approach for the de-interleave and interleave operations at the source and display where the entire image is being manipulated. A pass through example of this can be found in the blkproc_ex.mdl model. This example demonstrated gains of about 8% in resource utilization.

A second source of optimization can come from choosing a template ROI that is 8x8 or 16x16 and utilizing the TI C64x+ core IMGLIB with the DMA block processing utility to recreate the SAD algorithm. However, since the TFL already uses hardware intrinsics for processing, the gains may be minimal. An example model of this (requires the TI IMGLIB) can be found in the imglib_ex.mdl.

Action We can now generate, build, load, and execute code on the processor. Let the application run for a few seconds, and then halt the processor and view the profiling results.

Note: *For full performance, we may still want to incorporate the DMA and IMGLIB SAD approaches described above.*



Profiled Simulink Subsystems

System name	SAD_stabilize_final_rowmajor_fixpt_vpfeoptimized/Stabilization Algorithm
STS object	stsSys5_OutputUpdate
Maximum time spent in this subsystem	60.57 ms (363% of base interval)
Average time spent in this subsystem	46.47 ms (278% of base interval)
Number of iterations counted	4464

System name	SAD_stabilize_final_rowmajor_fixpt_vpfeoptimized/Stabilization Algorithm/Stabilization/Estimate Motion/SAD
STS object	stsSys2_OutputUpdate
Maximum time spent in this subsystem	29.04 ms (174% of base interval)
Average time spent in this subsystem	29.02 ms (174% of base interval)
Number of iterations counted	4464

Finally, we can run the stack profiler to determine the application's stack usage. To do this, you can type the following at the MATLAB command prompt:

Action

```
>> profile(CCS_Obj,'stack','setup')
```

This fills the processor's stack with a known repeating value, and displays the initial stack usage which is 0%.

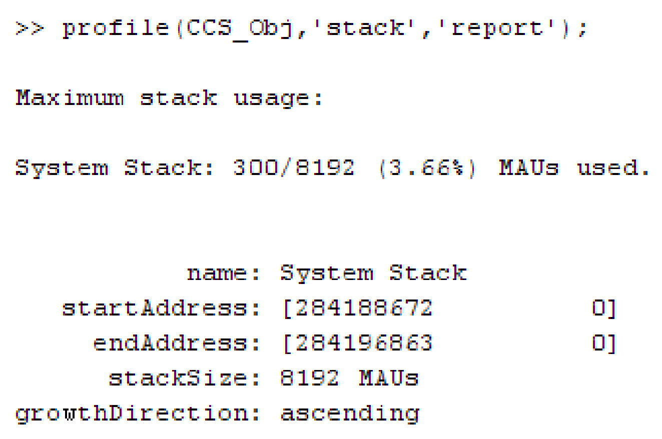
Action Next, re-run the application in Code Composer Studio. After some time has passed, halt the application.

We can get a profile of the stack usage by entering the following command on the MATLAB prompt:

Action

```
>> profile(CCS_Obj,'stack','report')
```

Given the results, you can either decrease or increase the size you allocated for the stack. You can update the stack size by going to **Simulation** > **Configuration Parameters...** > **Embedded IDE Link CC**. Adjust the value written under System stack size (bytes).



```
>> profile(CCS_Obj,'stack','report');

Maximum stack usage:

System Stack: 300/8192 (3.66%) MAUs used.


           name: System Stack
   startAddress: [284188672          0]
     endAddress: [284196863          0]
      stackSize: 8192 MAUs
growthDirection: ascending
```

This concludes Experiment 2

This concludes Lab 2

Appendix A - Peripheral Device and Hardware Modeling

Tasks:

- Row major data organization
- Video source and output modeling

Details:

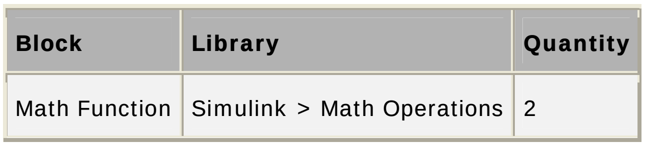
Row major data organization

Simulink has column-major organization so that it can interface with MATLAB and the Fortran BLAS libraries of the PC. This speeds up PC simulation, but the peripheral devices for the target hardware have row-major data organization which will require an expensive image transpose without modifying the underlying model.

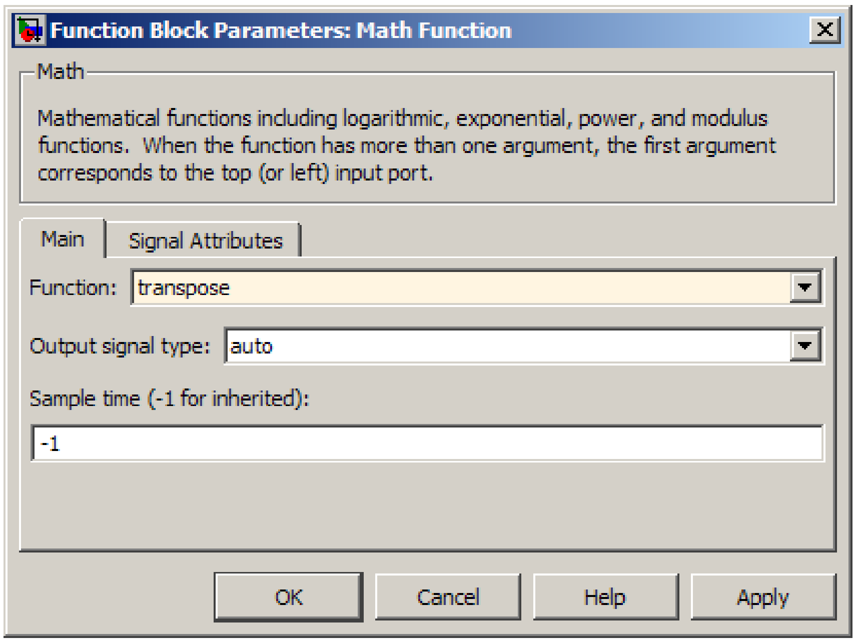
The simplest way to modify our model with row-major organization is to add a transpose block to the input source and displays. This changes our input data to be row-major and we can now modify the parameters in the model to work with this data.

Open the SAD_stabilize_final_fixpt.mdl model.

Add a transpose to the source and display for the model. The transpose operation is part of the Math Function block in Simulink.



Block	Library	Quantity
Math Function	Simulink > Math Operations	2



Mathematical functions including logarithmic, exponential, power, and modulus functions. When the function has more than one argument, the first argument corresponds to the top (or left) input port.

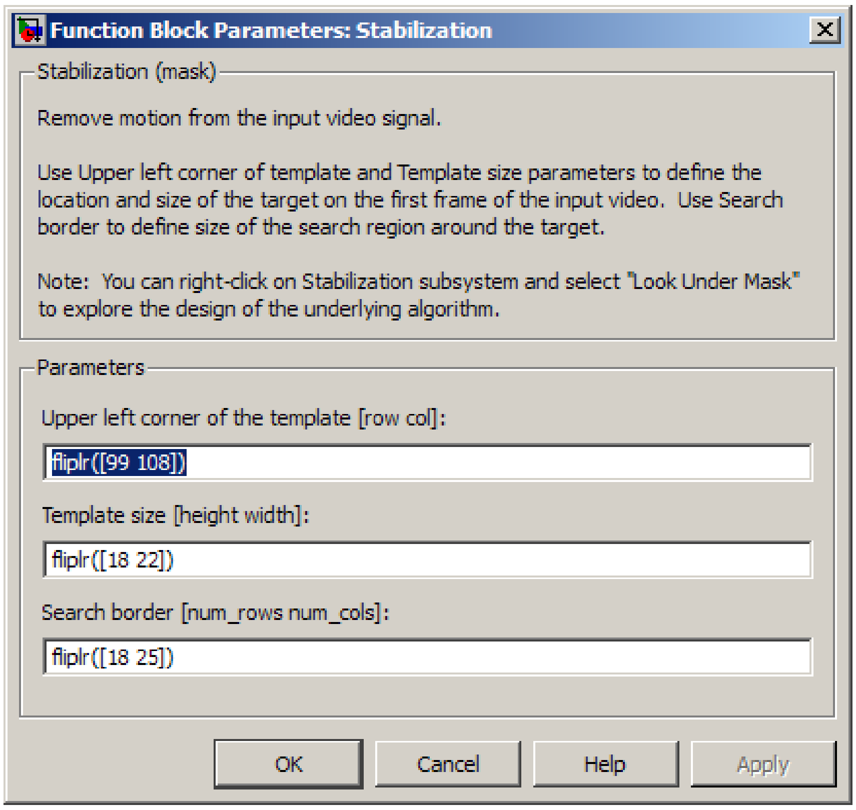
Function: transpose

Output signal type: auto

Sample time (-1 for inherited):

-1

Since our model has been parameterized with the ROI search region coordinates and sizes, the row-major conversion is as simple as flipping the coordinates for the sizes. We can use the `fliplr` MATLAB command or simply change the order of the values for each parameter.



Function Block Parameters: Stabilization

Stabilization (mask)

Remove motion from the input video signal.

Use Upper left corner of template and Template size parameters to define the location and size of the target on the first frame of the input video. Use Search border to define size of the search region around the target.

Note: You can right-click on Stabilization subsystem and select "Look Under Mask" to explore the design of the underlying algorithm.

Parameters

Upper left corner of the template [row col]:

fliplr([99 108])

Template size [height width]:

fliplr([18 22])

Search border [num_rows num_cols]:

fliplr([18 25])

Save the model as SAD_stabilize_final_rowmajor_fixpt.mdl.

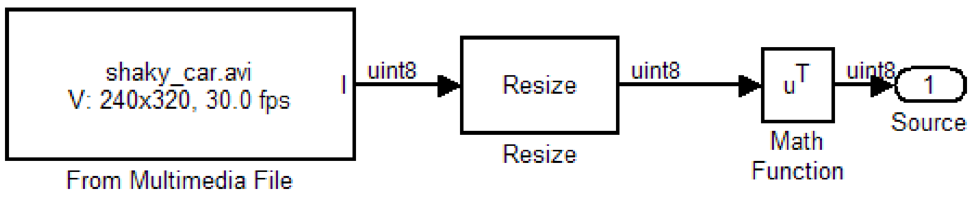
Video source and output modeling

To complete our model, we need to modify the input source such that the video size is the same as what we will be getting from the peripheral device on our target hardware. Additionally, we need to determine what the display will be and build up the model appropriately.

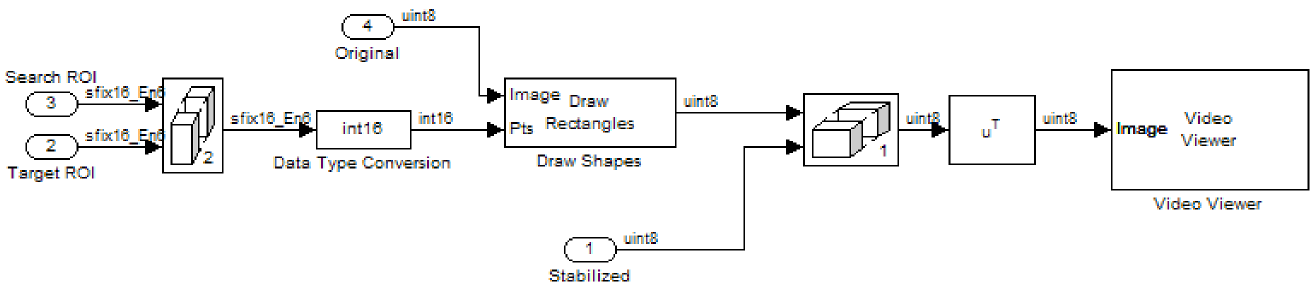
Our target hardware is utilizing a VGA input, which has a size of 600 lines and 800 columns (800x600). We can modify our input source to utilize a video of this size and adjust our settings appropriately. Another option is to resize the input to our stabilization algorithm such that we process a smaller amount of data. The Video Processing Front End (VPFE) of the DM6437 has a hardware scaler that we can utilize.

For display, we can use the Video Processing Back End (VPBE) and output to the LCD. The output can be the smaller input image or a re-scaled version of the smaller image. Another possibility is to scale the motion vectors based on the resize in the front end and apply the scaled vectors to the full size input for display. We may also want to draw our regions of interest and target image in the output as well as the un-stabilized input. Each of these has possibilities for hardware acceleration with the OSD.

Let’s start with the input image. The input is 800x600 and we can resize it to match ¼ of a composite video input. This makes our input image (320x240) or equivalent to the input video size we started with. To model a resize from the hardware, we can insert a resize block in our row major model and update the starting location and size of the template/search region; however since our input video is already sized correctly, the resize will just be a pass through. If the input video were actually 800x600 then we would need to adjust the parameters for the block and tune the template starting location.



The display for our new model will add the target ROI and search region ROI in the original unstabilized image. We will concatenate this result with the stabilized image for final viewing.



The updated model is SAD_stabilize_final_rowmajor_fixpt_hw.mdl. Simulate the results of this model to observe the behavior of the hardware.

Note: *Over time we may start to see drift in the region of interest. This can be corrected with an error metric and state reset.*

Appendix B - Generate and Verify Code w/ P-I-L

Tasks

- Add Target Preferences Block
- Configure Target Preferences Block
- Configure Real-Time Workshop Parameters
- Build the Model
- Configure the model for PIL
- Build & Verify PIL Block

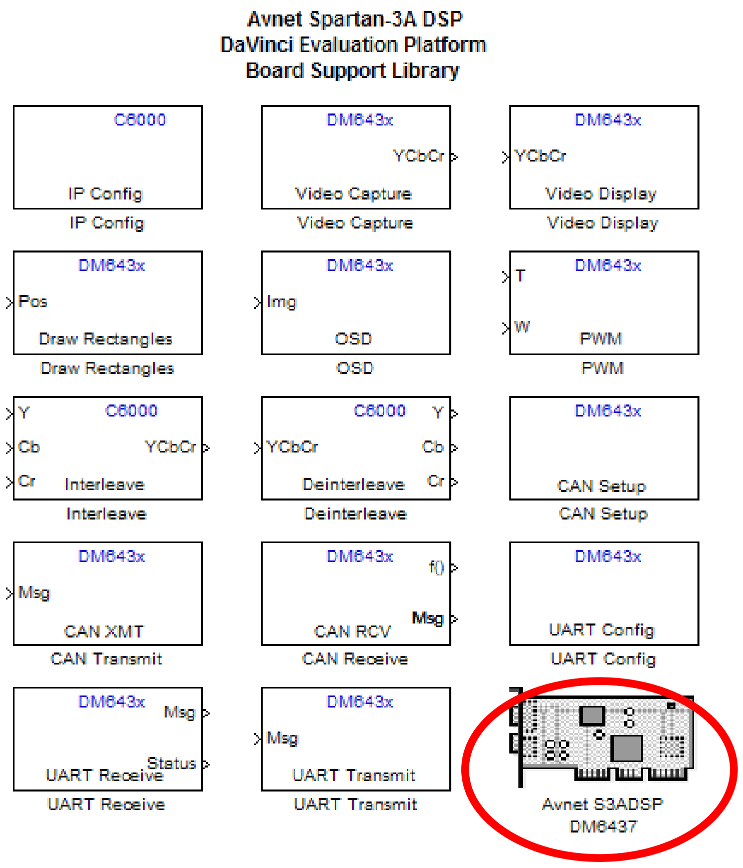
Details:

Add Target Preferences Block

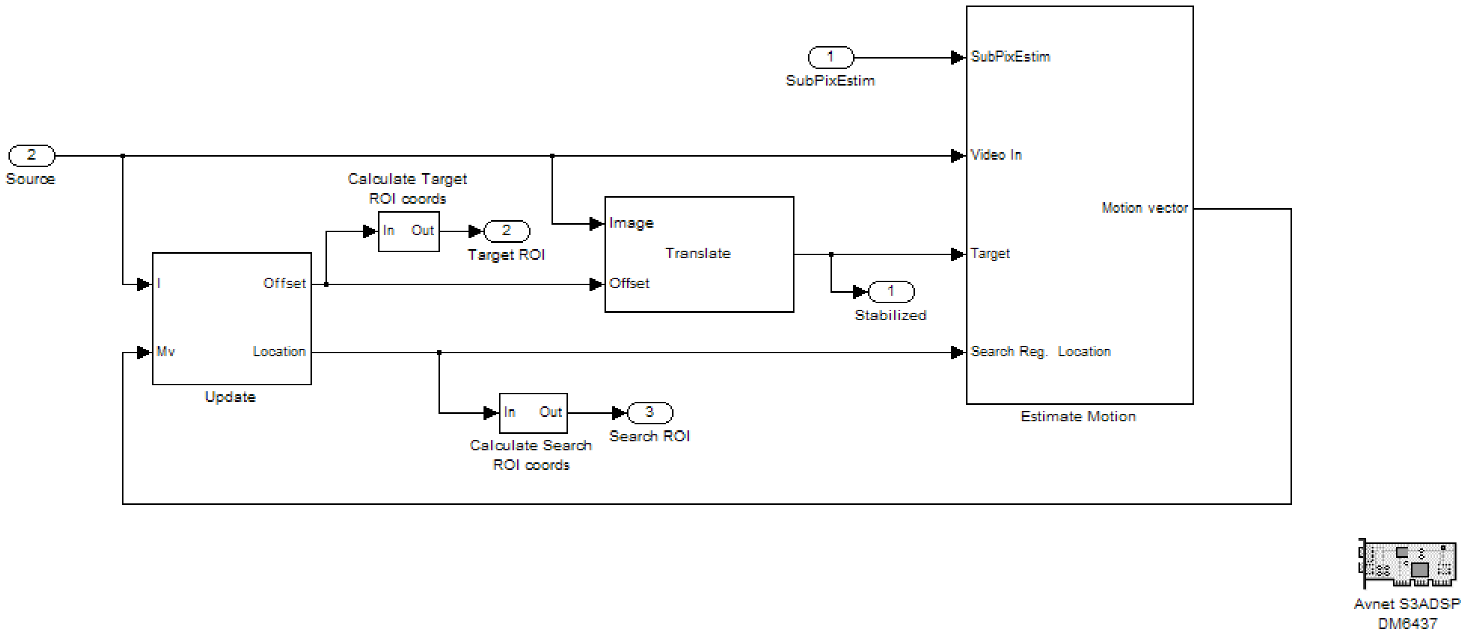
All models require a Target Preferences block to generate code specific to TI processors. It enables you choose a TI processor, define a memory map, and allocate code and data memory sections.

Note: *Every model must have exactly one Target Preferences block.*

Open the avnet_s3adsp_dm6437 library and locate the Target Preferences block.

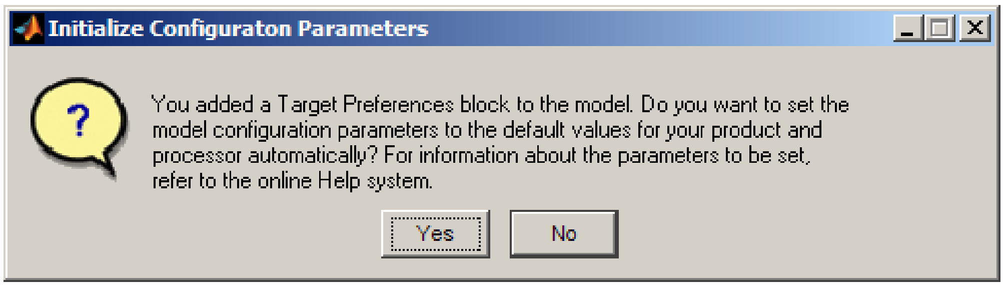

Avnet Spartan-3A DSP
DaVinci Evaluation Platform
Board Support Library
C6000
IP Config
IP Config
DM643x
YCbCr
Video Capture
Video Capture
DM643x
YCbCr
Video Display
Video Display
DM643x
Pos
Draw Rectangles
Draw Rectangles
DM643x
Img
OSD
OSD
T
W
DM643x
PWM
PWM
Y
Cb
Cr
C6000
YCbCr
Interleave
Interleave
C6000
YCbCr
Y
Cb
Cr
Deinterleave
Deinterleave
DM643x
CAN Setup
CAN Setup
DM643x
Msg
CAN XMT
CAN Transmit
DM643x
f()
Msg
CAN RCV
CAN Receive
DM643x
UART Config
UART Config
DM643x
Msg
Status
UART Receive
UART Receive
DM643x
Msg
UART Transmit
UART Transmit
Avnet S3ADSP
DM6437

Copy the Target Preferences block inside the mask of the stabilization subsystem.



Click **Yes** on the dialog box that asks if you want to initialize the simulation parameters to the target's default settings.

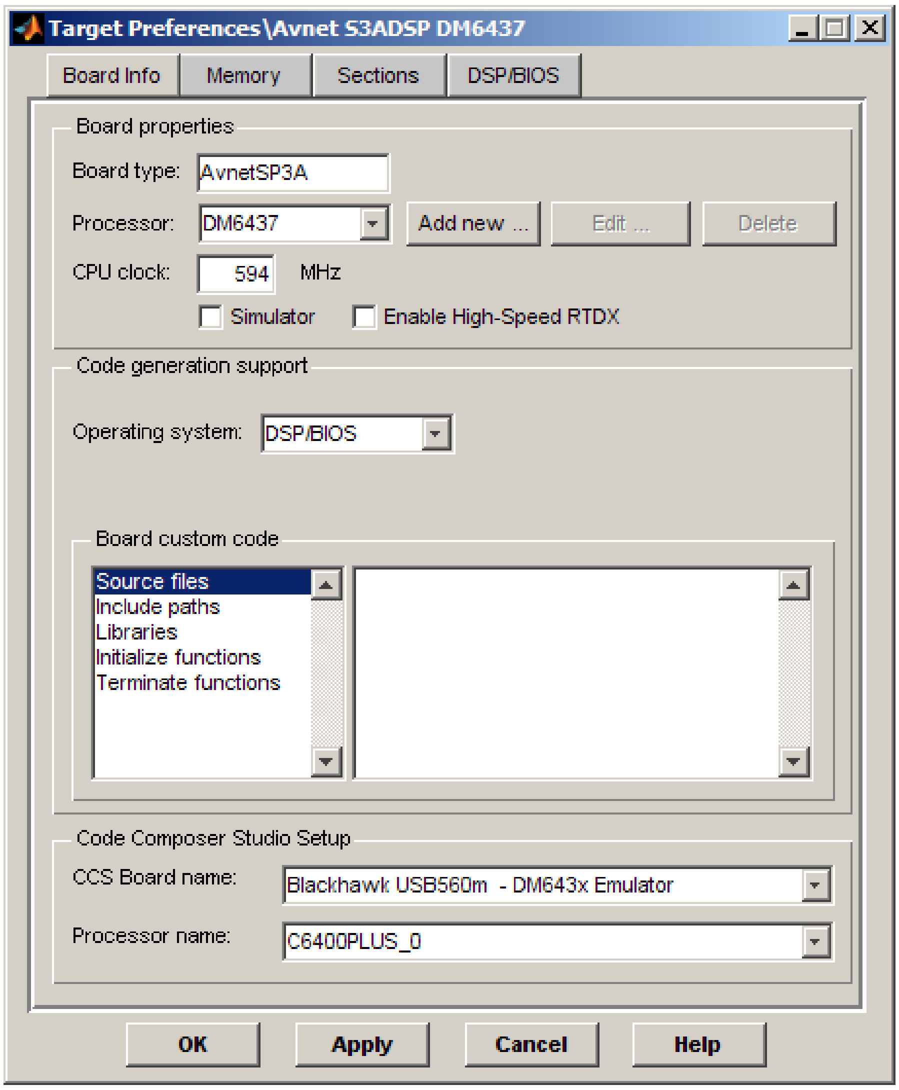
Note: *This will configure most of the model settings for us and we can review them later.*


Initialize Configuraton Parameters

You added a Target Preferences block to the model. Do you want to set the model configuration parameters to the default values for your product and processor automatically? For information about the parameters to be set, refer to the online Help system.

Configure Target Preferences Block

Double click on the Target Preferences block. (click ok on the pop-up)



Target Preferences\Avnet S3ADSP DM6437

Board properties

Board type: AvnetSP3A

Processor: DM6437

CPU clock: 594 MHz

☐ Simulator ☐ Enable High-Speed RTDX

Code generation support

Operating system: DSP/BIOS

Board custom code

Source files
Include paths
Libraries
Initialize functions
Terminate functions

Code Composer Studio Setup

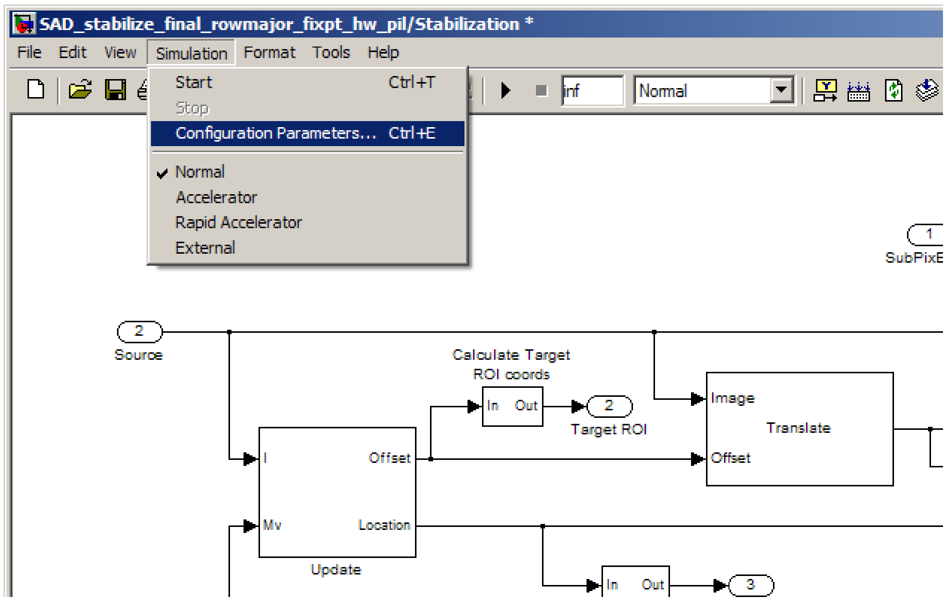
CCS Board name: Blackhawk USB560m - DM643x Emulator

Processor name: C6400PLUS_0

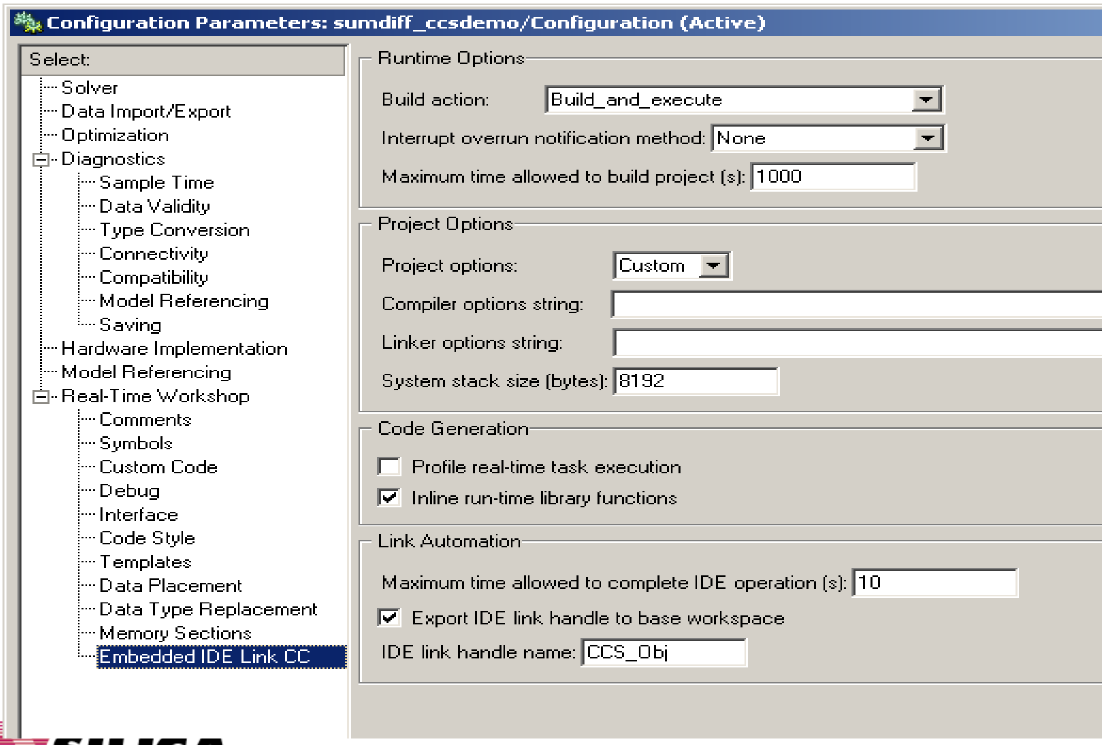
Set the Operating System to None and click OK to save the settings.

Configure Real-Time Workshop Parameters

From the model menu bar, select **Simulation > Configuration Parameters**.



Click Embedded IDE Link CC entry under Real-Time Workshop in the left pane.



Runtime Options

Build action: Build_and_execute

Interrupt overrun notification method: None

Maximum time allowed to build project (s): 1000

Project Options

Project options: Custom

Compiler options string:

Linker options string:

System stack size (bytes): 8192

Code Generation

☐ Profile real-time task execution

☑ Inline run-time library functions

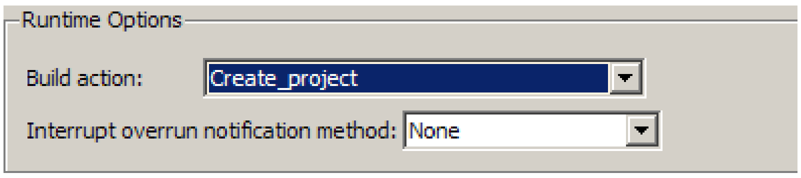
Link Automation

Maximum time allowed to complete IDE operation (s): 10

☑ Export IDE link handle to base workspace

IDE link handle name: CCS_Obj

Change the Build Action to Create Project.


Runtime Options

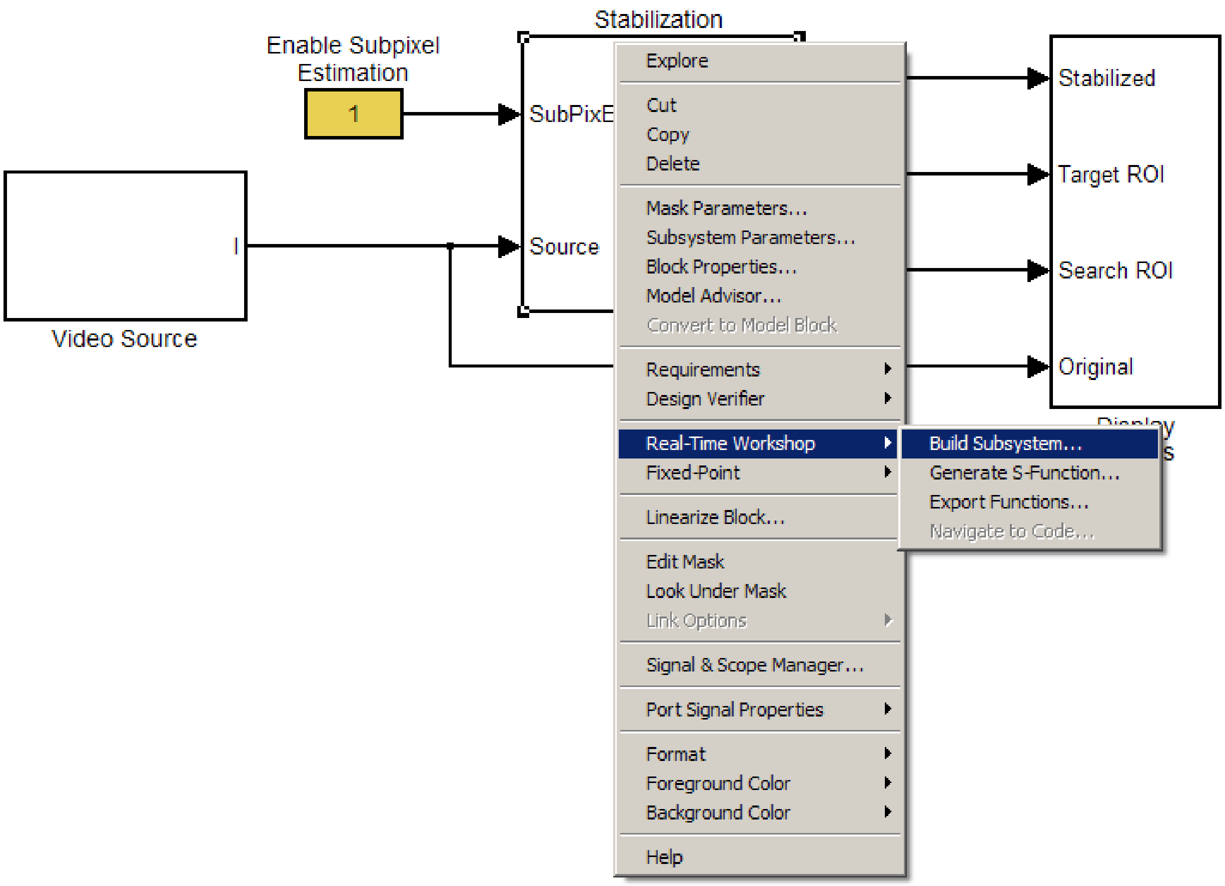
Build action: Create_project

Interrupt overrun notification method: None

Select OK.

Build the Model

Build the stabilization subsystem by right-clicking the block and selecting **Real-Time Workshop > Build Subsystem.**

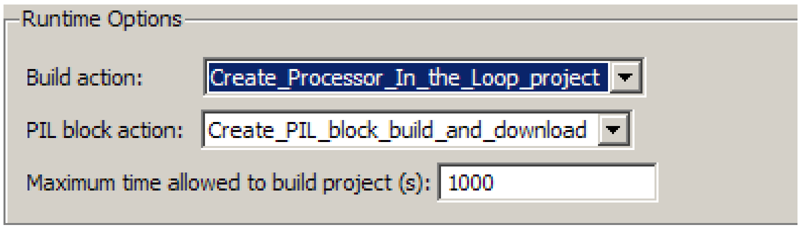


You can navigate through the code inside the created Code Composer Studio project.

Configure the model for PIL

Open the Configuration Parameters again. Click Embedded IDE Link CC entry under Real-Time Workshop in the left pane.

Set the Build action to Create_Processor_In_the_Loop_project and set PIL block action to Create_PIL_block_build_and_download.


Runtime Options

Build action: Create_Processor_In_the_Loop_project

PIL block action: Create_PIL_block_build_and_download

Maximum time allowed to build project (s): 1000

Click OK to save the settings.

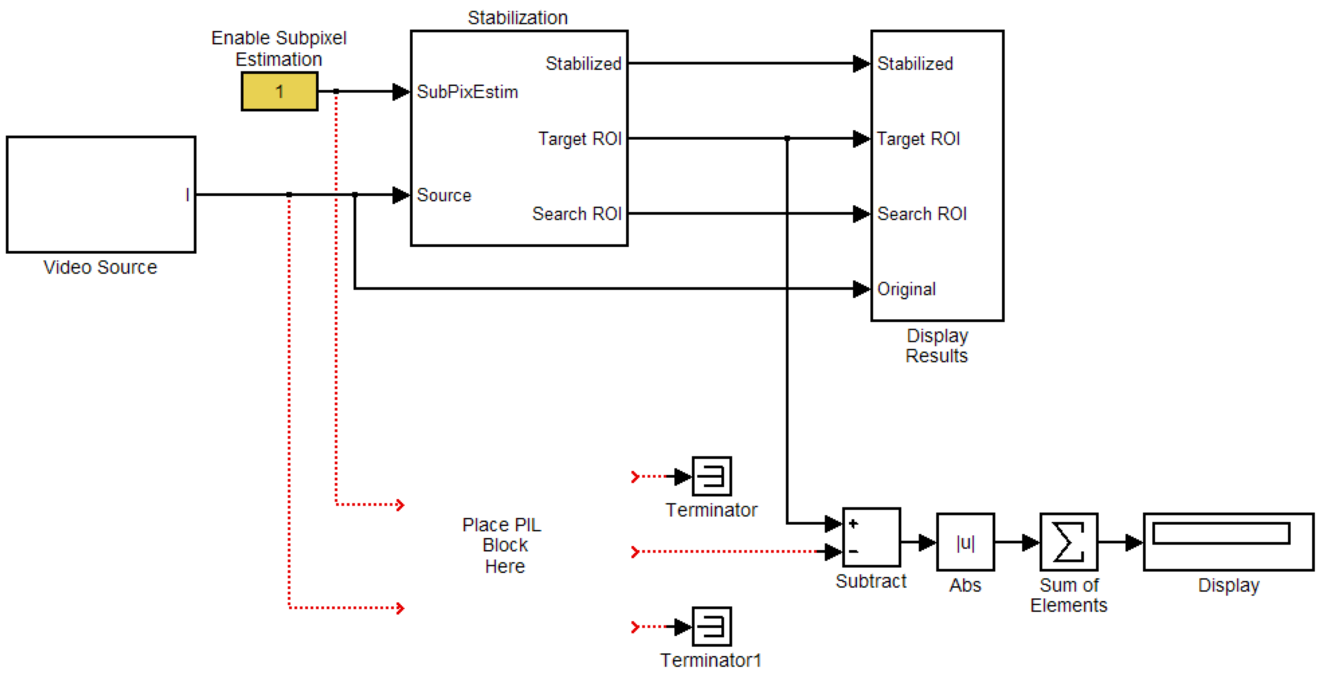
These steps configured the model so that when you generate code for the "Stabilization" subsystem, the build process:

- Creates a PIL block in a new model
- Generates the PIL algorithm code and project in the CCS window
- Builds the generated project
- Loads the program onto the processor.

Build & Verify PIL Block

Build the stabilization subsystem by right-clicking the block and selecting **Real-Time Workshop > Build Subsystem.**

Copy the PIL block to the model.



Simulate the model and verify that the results for any port are zero.

Note: *PIL automates the debugger in Code Composer Studio and passes data from Simulink to the processor via JTAG. This results in a very slow simulation, but it allows you to verify frames of data.*

Revision History

Date	Version	Revision
19/11/08	.1	Merged Lab #2 models with lab 5 models.
23/11/08	.2	Need to verify in hardware
23/11/08	1.0	Mario verified models in HW
16/12/08	1.1	Updated for Archive
12/1/09	1.2	Updated for patch 1.05b
21/1/09	3.0	Version to match Avnet, Created Appendix, Experiment End Markers