INTRODUCTION

The Square-Law Detector is one of the methods used for demodulating Amplitude Modulated (AM) signals.

The primary goal of this technique is to retrieve the original message signal from the modulated carrier wave.

An AM signal can be expressed mathematically as:

$$s(t) = A_c[1 + k_a m(t)]\cos\omega_c t$$

where:

- Ac is the amplitude of the carrier wave,
- Ka is the modulation index,
- m(t) is the message signal,
- fc is the carrier frequency.

The modulated signal consists of a carrier wave whose amplitude varies according to the message signal.

The Square-Law Detector operates on the principle that when an AM signal is squared, it produces components at both low and high frequencies. When we square the AM signal, we get:

$$s^2(t) = A_c^2[1 + k_a m(t)]^2\cos^2\omega_c t = 0.5A_c^2[1 + k_a m(t)]^2 + 0.5A_c^2[1 + k_a m(t)]^2\cos 2\omega_c t$$

This equation shows that squaring the AM signal results in three components:

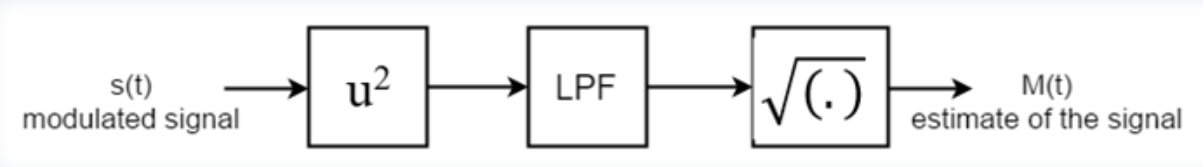
- A constant term,
- A term proportional to m(t),
- A high-frequency component at 4fc

To recover the original message signal m(t), we need to eliminate the high-frequency components introduced during squaring. This is typically done using a low-pass filter (LPF). The LPF allows only low-frequency signals to pass through while attenuating higher frequencies.

After passing through a low-pass filter and getting the square root, we obtain:

$$M(t) = K_a \cdot m(t) + \text{noise}$$

Ka is a constant related to system gain and noise represents any unwanted signals that may have been introduced during processing.



the Square-Law Detector technique for AM demodulation effectively retrieves information from an amplitude-modulated signal by squaring it and filtering out high-frequency components, allowing for recovery of the original message signal under appropriate conditions.

MBD AND MBSE

Model-Based Design (MBD) is a systematic approach that utilizes models throughout the development process of complex systems. It emphasizes the use of mathematical and graphical representations to design, simulate, and validate systems before physical implementation. This methodology allows engineers to visualize system behaviour, test various scenarios, and refine designs iteratively. MBD integrates modelling with simulation, automated testing, and code generation, which significantly enhances productivity and reduces time-to-market.

https://en.wikipedia.org/wiki/Model-based_design

Model-Based Systems Engineering (MBSE) is a formalized methodology that emphasizes the use of models as the primary means of information exchange, feedback, and requirements throughout the lifecycle of complex systems. Unlike traditional document-centric approaches, which rely heavily on various disconnected documents, MBSE integrates digital models to capture system requirements, design, analysis, verification, and validation processes.

https://insights.sei.cmu.edu/blog/introduction-model-based-systems-engineering-mbse/

Model-Based Systems Engineering represents a transformative shift in how complex systems are designed and managed by leveraging digital models for improved clarity, collaboration, risk management, and efficiency throughout their lifecycle.

SIMULINK

Simulink is a key tool within the Model-Based Design framework developed by MathWorks. It provides a graphical environment for modeling, simulating, and analyzing dynamic systems. Users can create block diagrams that represent system components and their

interactions, enabling them to simulate how these components behave over time. Simulink supports various applications across industries such as automotive, aerospace, communications, and more.

https://uk.mathworks.com/solutions/model-based-design.html

Advantages of Model-Based Design with Simulink ARE:

Shortened Development Cycles: By using models for simulation and validation early in the design process, teams can identify issues sooner. This leads to reduced development time, compared to traditional methods where coding precedes testing.

Automation of Key Steps: MBD allows for automation in reporting, coding, and verification processes. This reduces manual errors associated with human intervention and streamlines workflows.

Traceability: MBD creates a digital thread that connects requirements through system architecture down to component design and testing. This traceability ensures that all aspects of the project are aligned with initial specifications.

Support for Agile Development: The iterative nature of MBD aligns well with Agile methodologies by enabling continuous integration of feedback through simulations and automated tests.

Enhanced Collaboration: With a single source of truth provided by models, different teams (designers, testers, integrators) can work collaboratively without miscommunication or discrepancies in understanding the system's functionality.

Facilitation of Complex System Design: MBD is particularly beneficial for designing complex systems where interactions between components are critical. Simulink's visual representation helps engineers understand these interactions better than traditional textual programming approaches.

https://uk.mathworks.com/company/technical-articles/automating-the-implementation-of-software-defined-radios-at-northrop-grumman.html

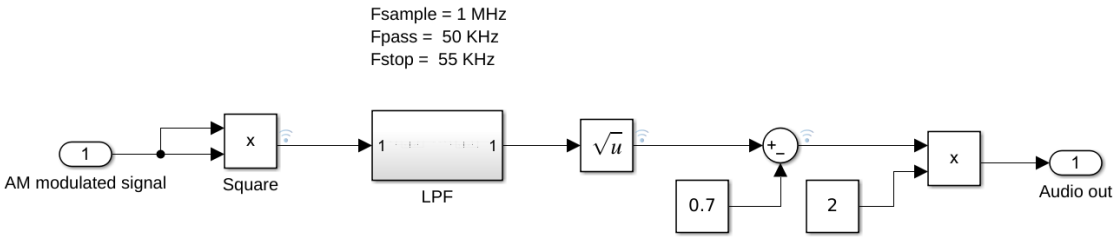
SIMULINK GOLDEN MODEL

A golden model in Simulink refers to a high-fidelity reference model that serves as the standard against which other models are validated.

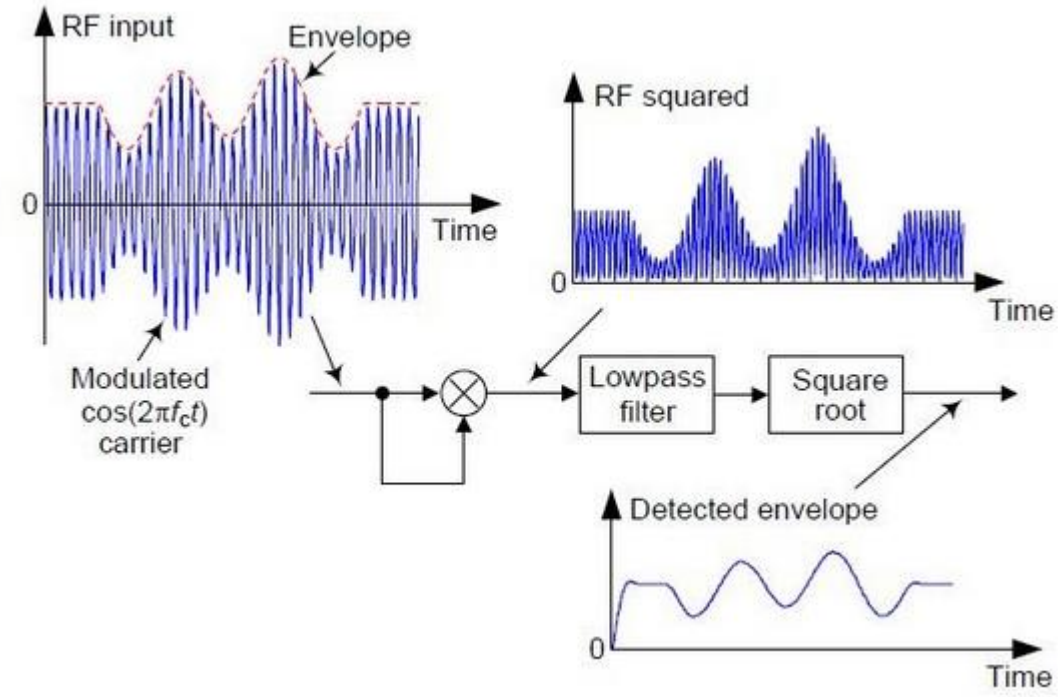
The golden model is typically characterized by its accuracy and completeness, ensuring that it represents the intended functionality of the system being modelled.

The primary purpose of a golden model is to provide a reliable benchmark for comparison during simulation and testing. This comparison can help identify discrepancies due to errors in modelling, coding mistakes, or unintended changes in system behaviour.

The golden model in Simulink:

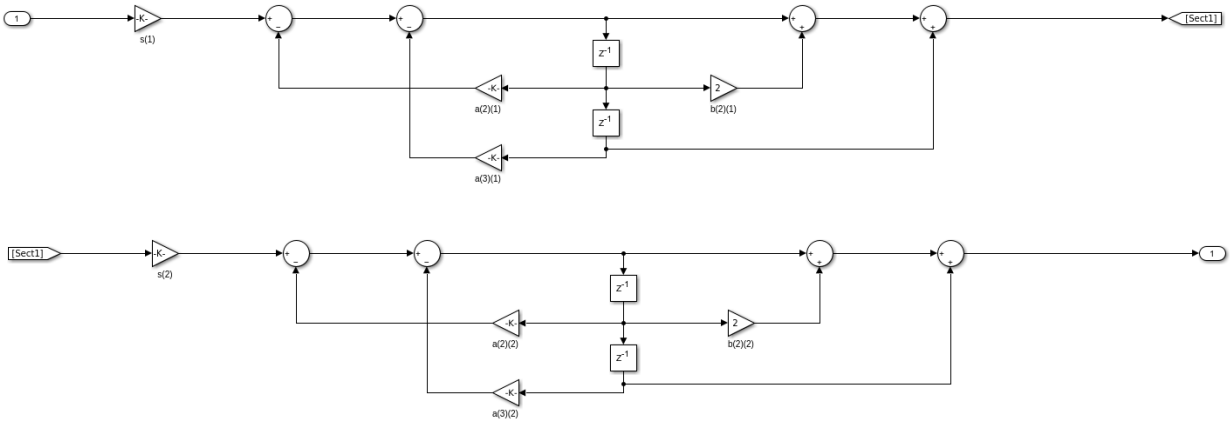


closely resembles the theoretical block diagram:

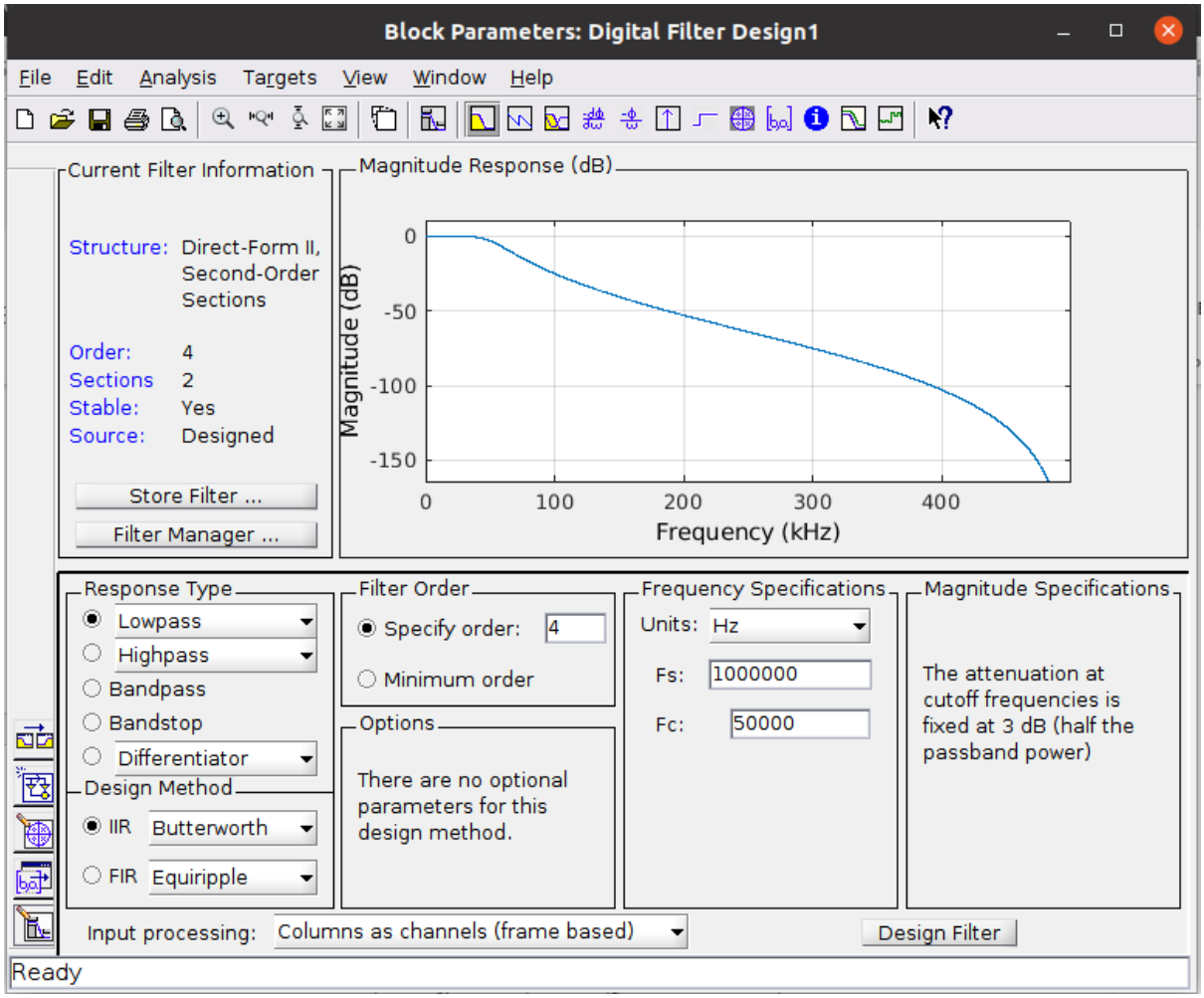


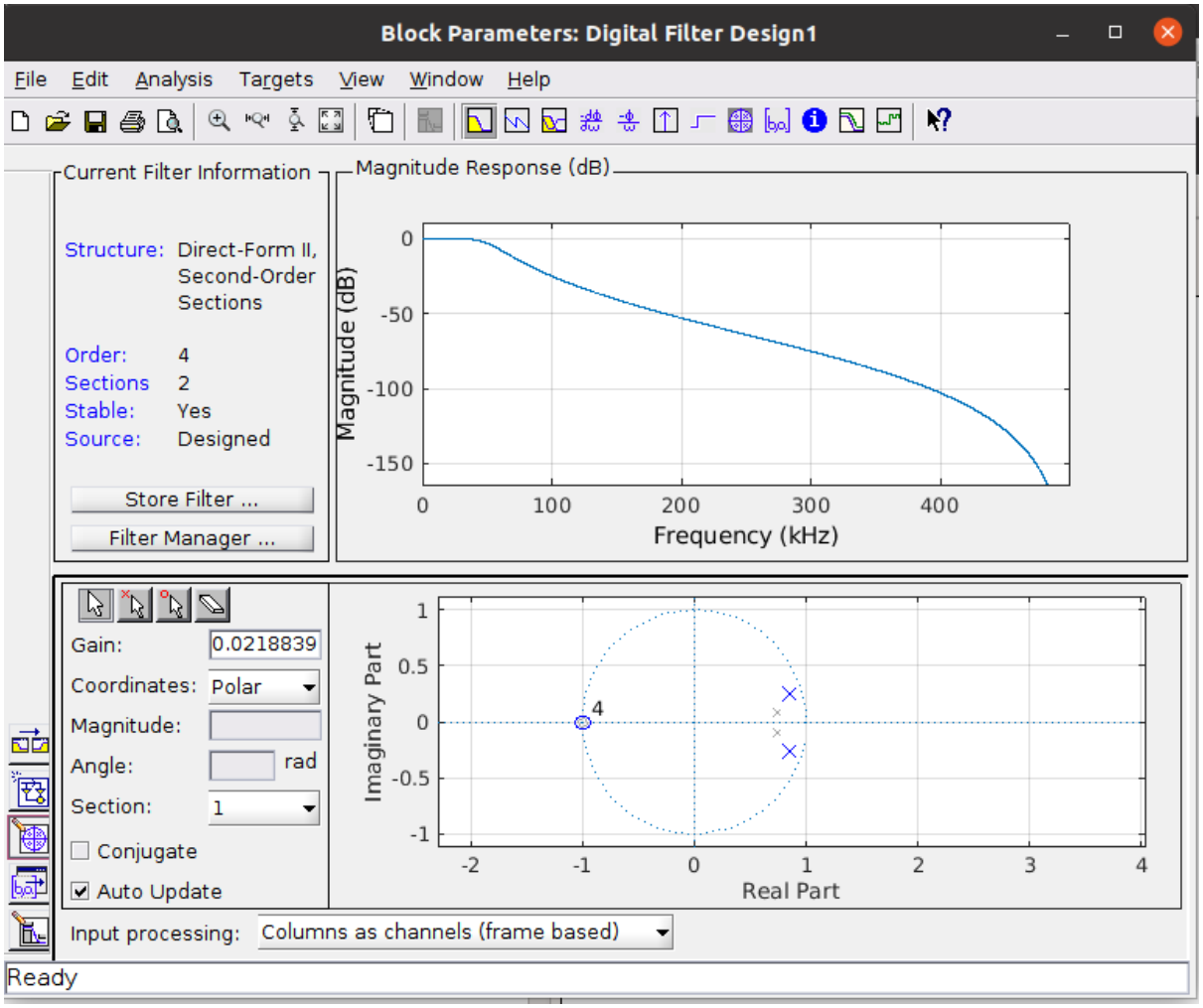
A DC offset remove block and some gain are the only additions.

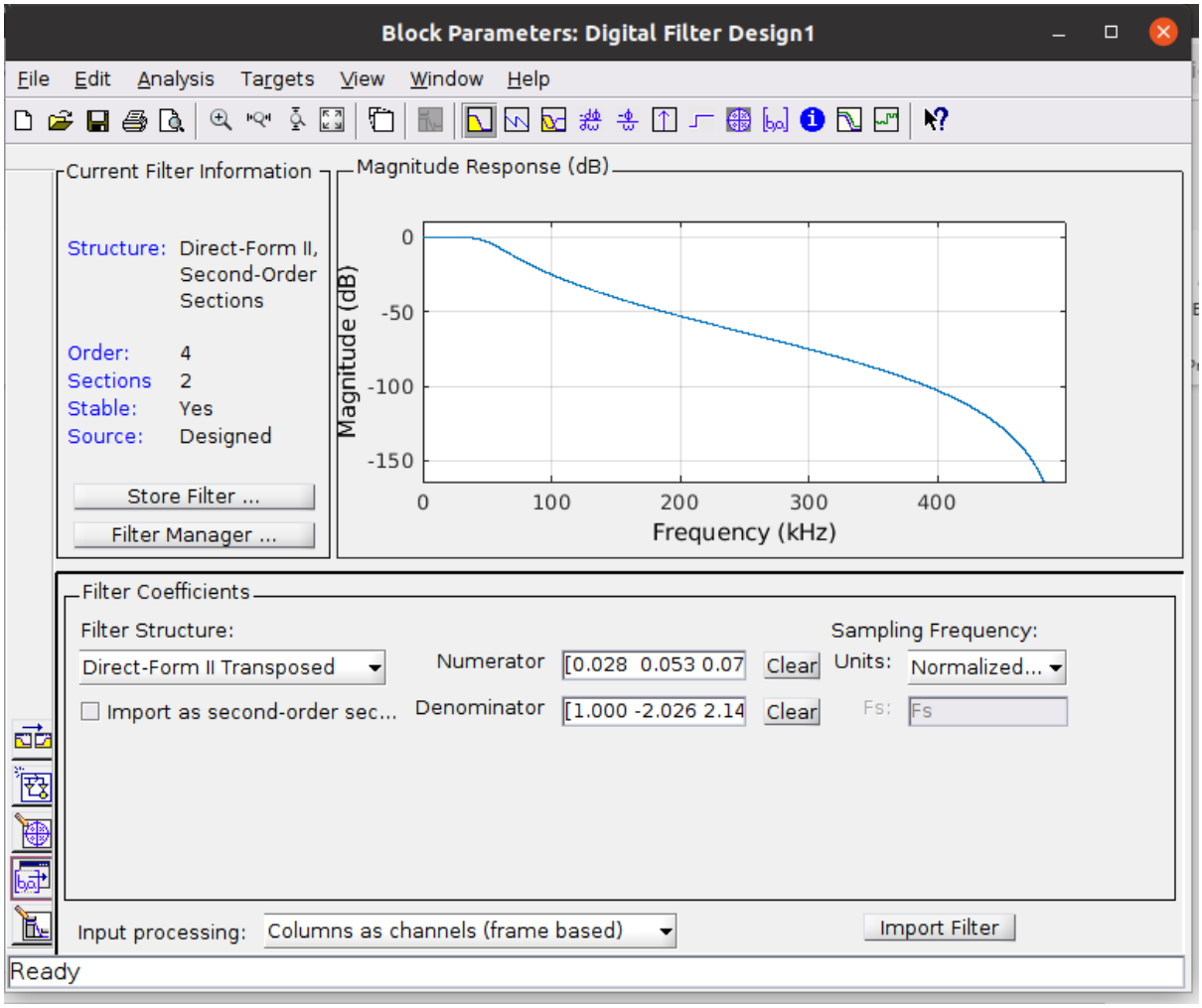
The LPF has been modelled as follows:



The filter has been designed in “Digital filter design”, from DSP system toolbox:




Block Parameters: Digital Filter Design1
File Edit Analysis Targets View Window Help
Current Filter Information
Structure: Direct-Form II, Second-Order Sections
Order: 4
Sections 2
Stable: Yes
Source: Designed
Store Filter ...
Filter Manager ...
Magnitude Response (dB)
Magnitude (dB)
0
-50
-100
-150
0
100
200
300
400
Frequency (kHz)
Gain: 0.0218839
Coordinates: Polar
Magnitude:
Angle: rad
Section: 1
Conjugate
Auto Update
Imaginary Part
1
0.5
0
-0.5
-1
-2
-1
0
1
2
3
4
Real Part
Input processing: Columns as channels (frame based)
Ready



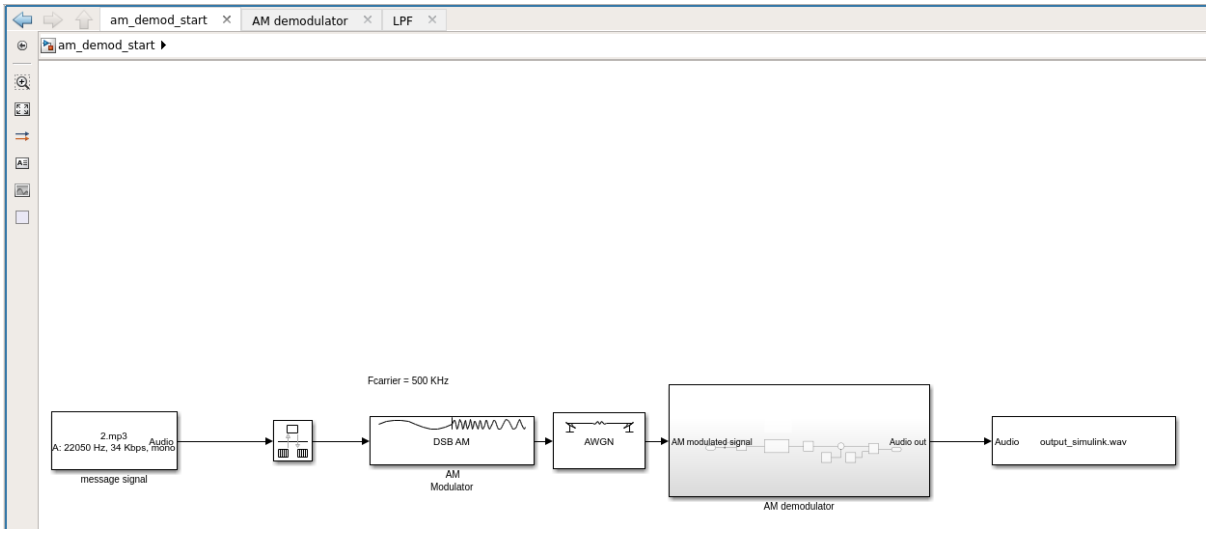
The DSP System Toolbox™ is a specialized software package designed for the design, simulation, and analysis of digital signal processing (DSP) systems. It is integrated with MATLAB® and Simulink®, providing a comprehensive environment for engineers and researchers to develop real-time signal processing applications across various domains such as communications, radar, audio processing, medical devices, and the Internet of Things (IoT).

https://uk.mathworks.com/products/dsp-system.html

The golden model has been exercised by real world ATC audio:

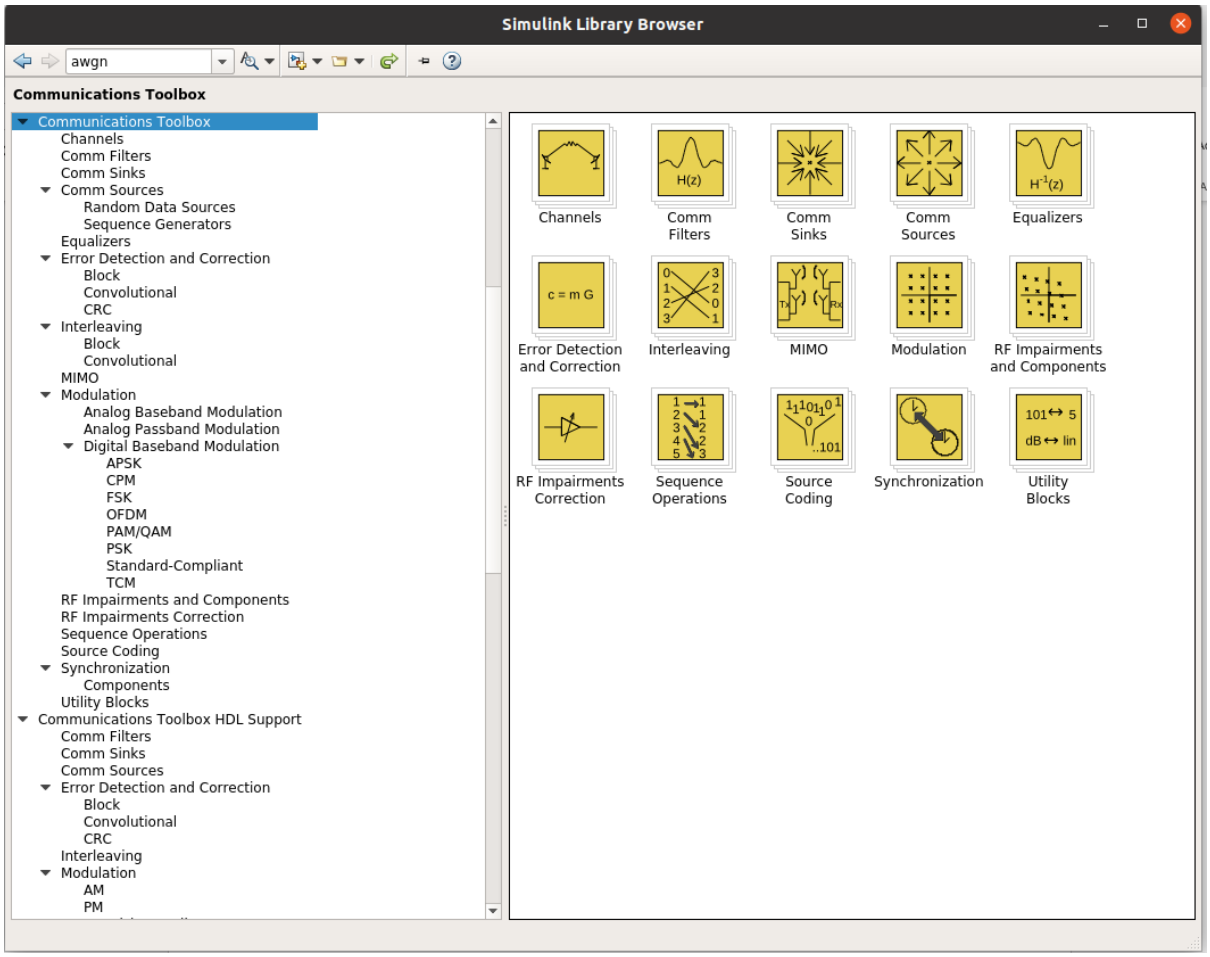
https://www.liveatc.net/recordings.php

As in Simulink is possible reading mp3 audio files:



The mp3 audio is fed to an AM modulator block, and the modulated signal is fed to the golden AM demodulator, whose output is saved on a wav output audio file.

The effect of an AWGN channel is added in the block diagram: this block comes from communication toolbox.

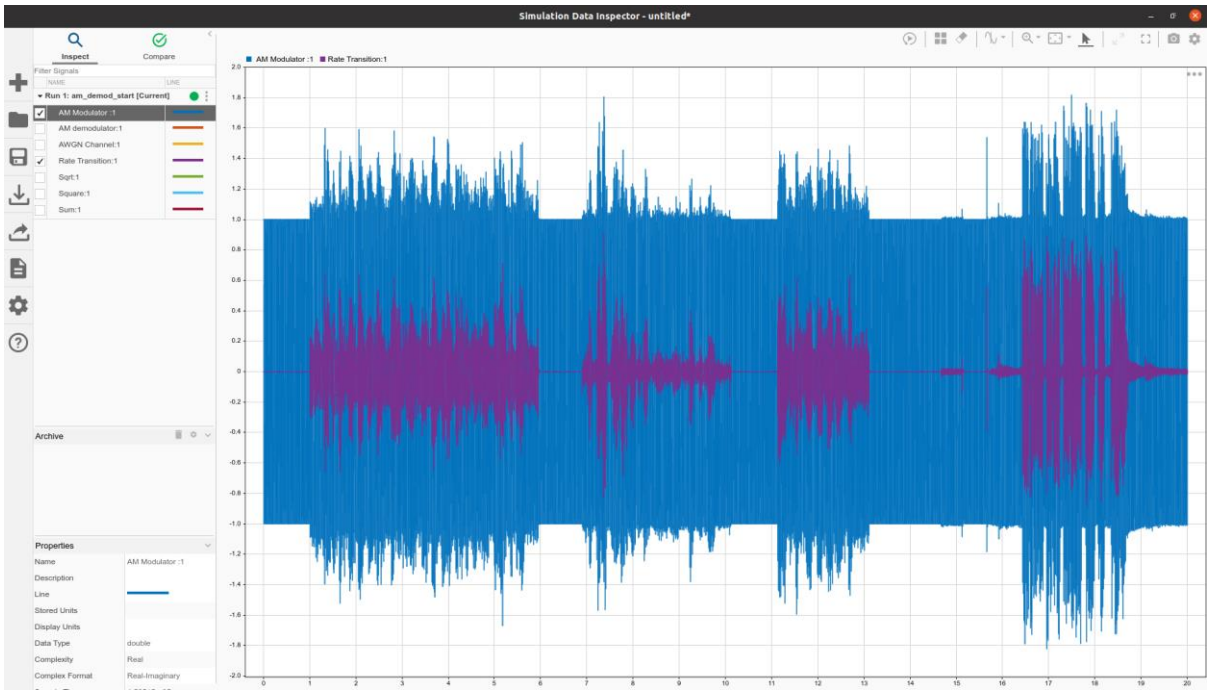


https://uk.mathworks.com/products/communications.html

The Communications Toolbox is a software suite designed for the design, simulation, analysis, and verification of communications systems. It provides a comprehensive set of algorithms and applications that facilitate the modelling and testing of various components within communication systems, particularly focusing on the physical layer.

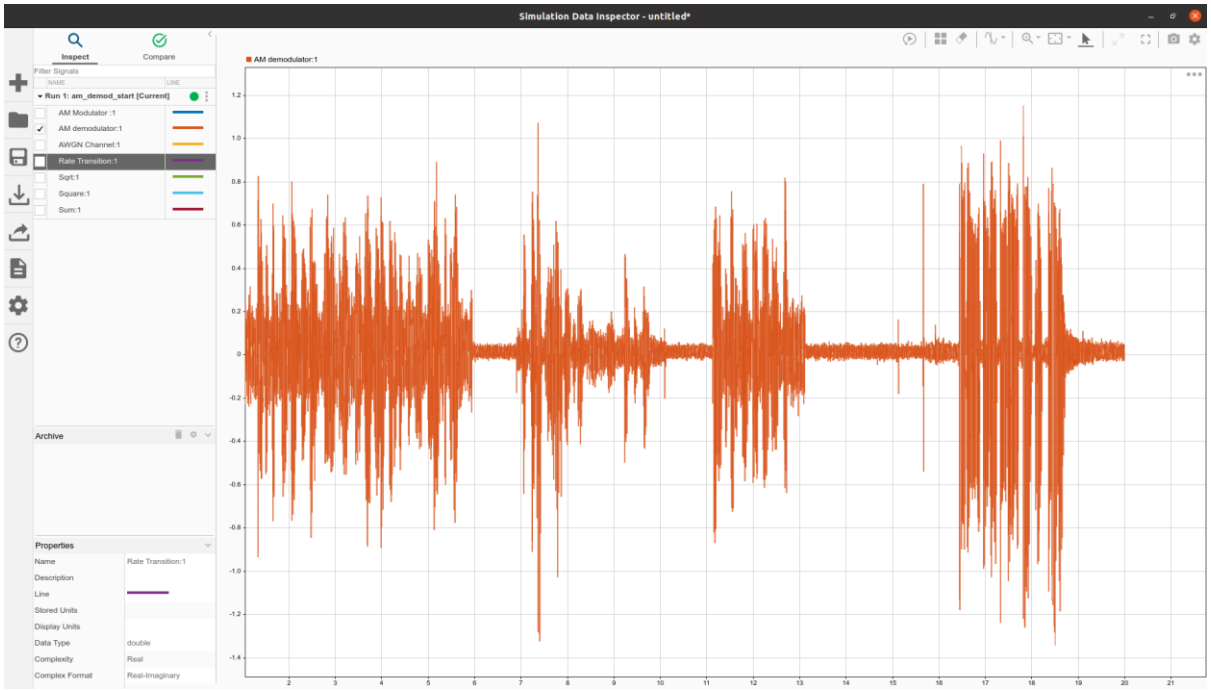
- Key Features are:
- Waveform Generation
- Receiver Performance Verification
- Artificial Intelligence Applications
- Propagation Channel Modeling
- Channel Degradation Compensation
- Software-Defined Radio (SDR) Testing
- Link-Level Simulation
- AI Techniques for Wireless Challenges
- Modeling RF Components
- Performance Acceleration
- Integration with Hardware: supports third-party SDR hardware like RTL-SDR and USRP radios, enabling practical implementation and testing of designs.

With the Simulink data inspector, waveforms at key points of the model can be seen in real time, as the model runs:

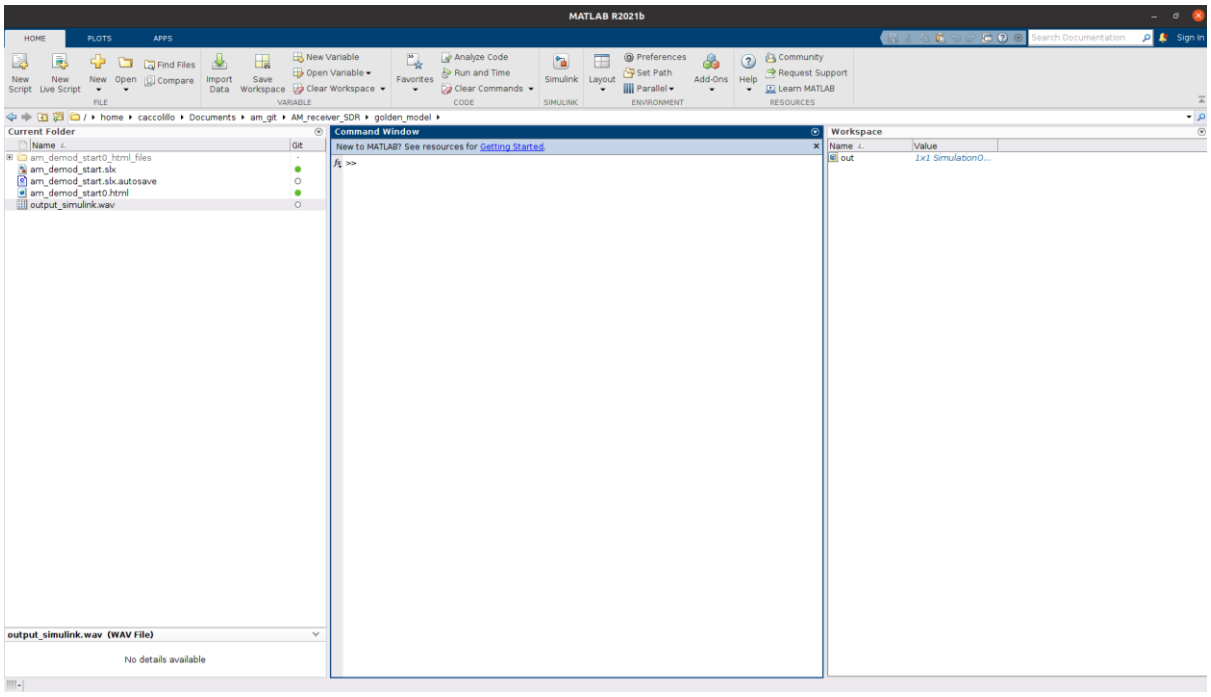


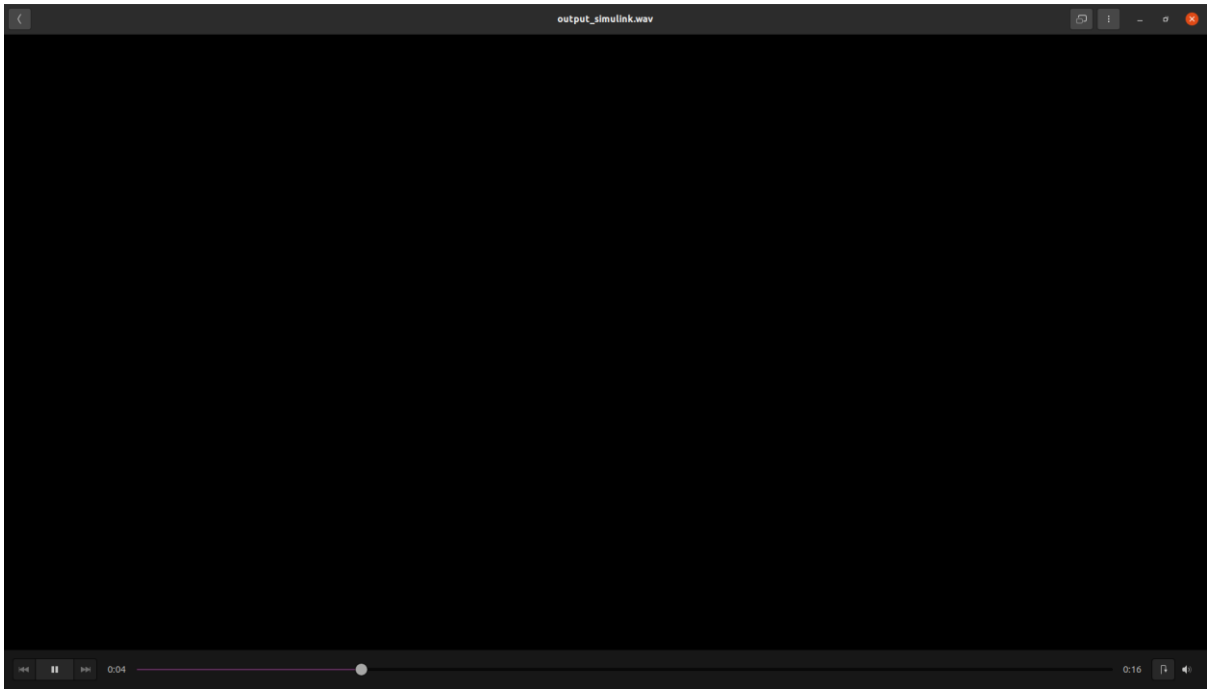
In purple there's the audio stream, and in blue there's the AM modulated input signal to the golden model.

The demodulated output audio signal can be viewed as well in the same manner:



Or can be listened too:



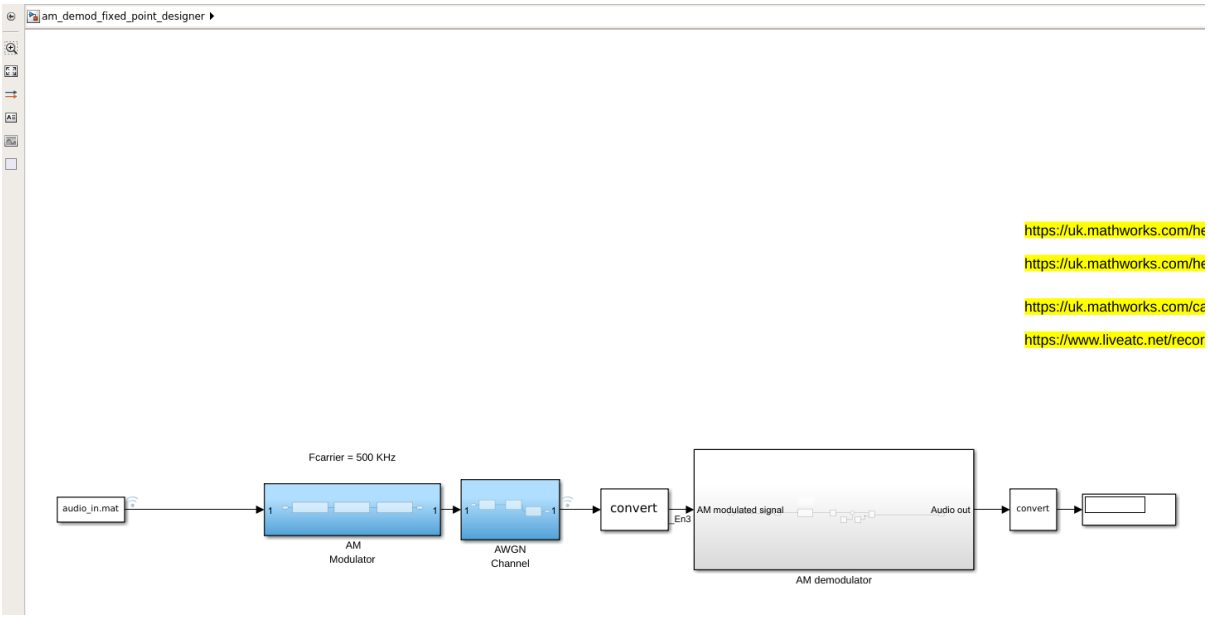


FIXED POINT CONVERSION

Fixed-Point Designer is a toolset for optimizing and implementing fixed-point and floating-point algorithms on embedded hardware. It offers specialized data types and numeric settings, enabling bit-true simulation for fixed-point systems to test and debug quantization effects like overflows and precision loss before hardware implementation. The tool provides apps for analyzing and converting double-precision algorithms to reduced-precision formats, helping users select data types that balance numerical accuracy and hardware constraints. It also allows replacing computationally expensive constructs with optimized hardware patterns. Additionally, it supports converting machine learning model parameters to fixed-point types and generating production-ready C and HDL code from optimized models.

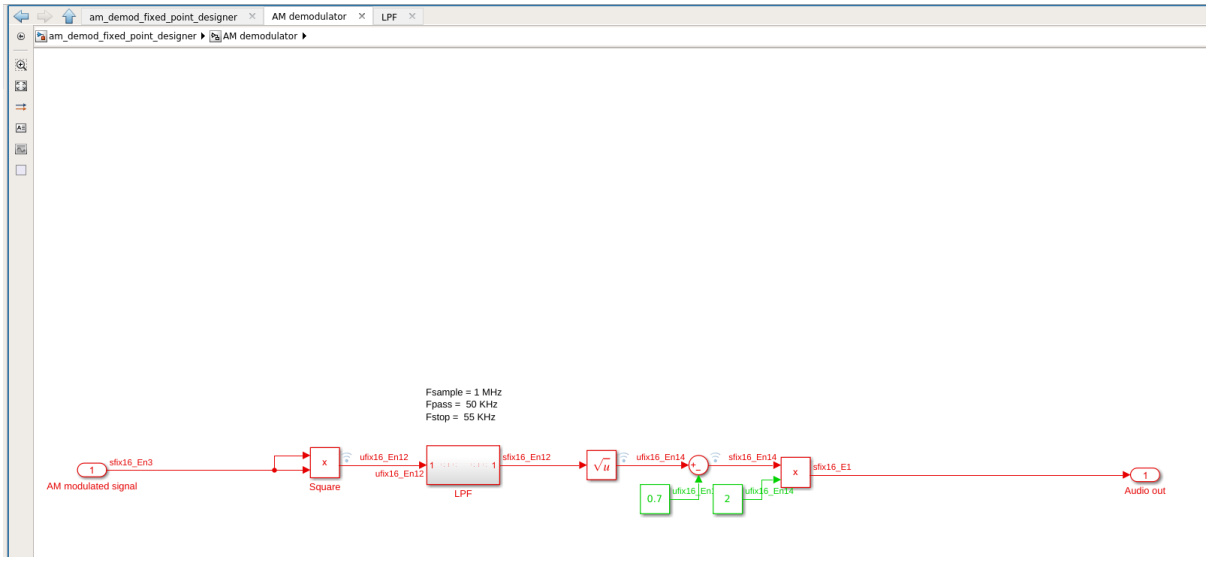
https://uk.mathworks.com/products/fixed-point-designer.html

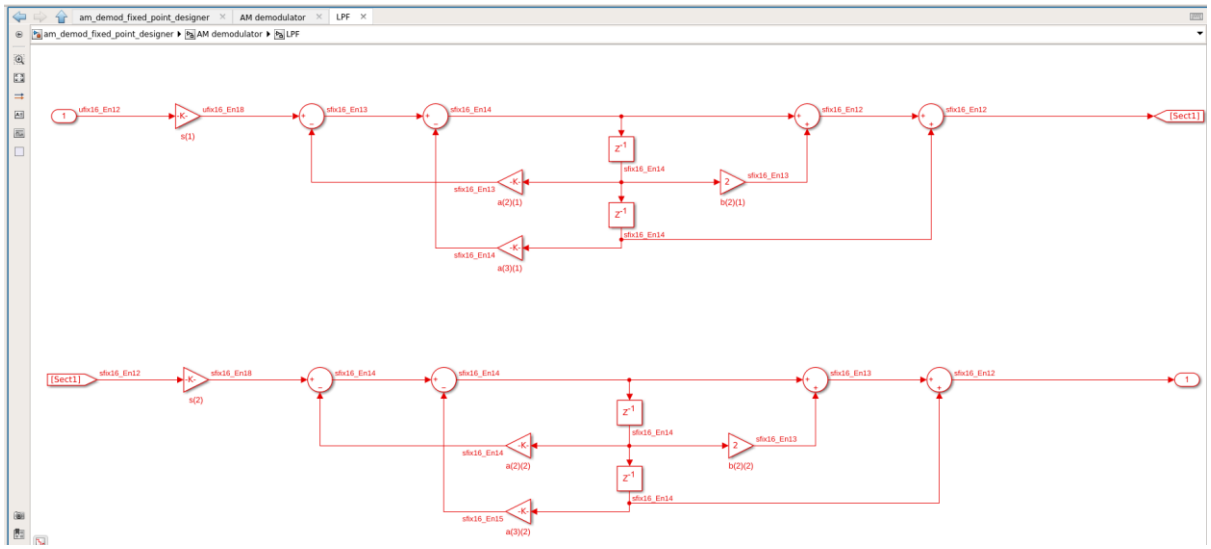
The floating point golden model, has been converted in fixed point thanks to this tool.



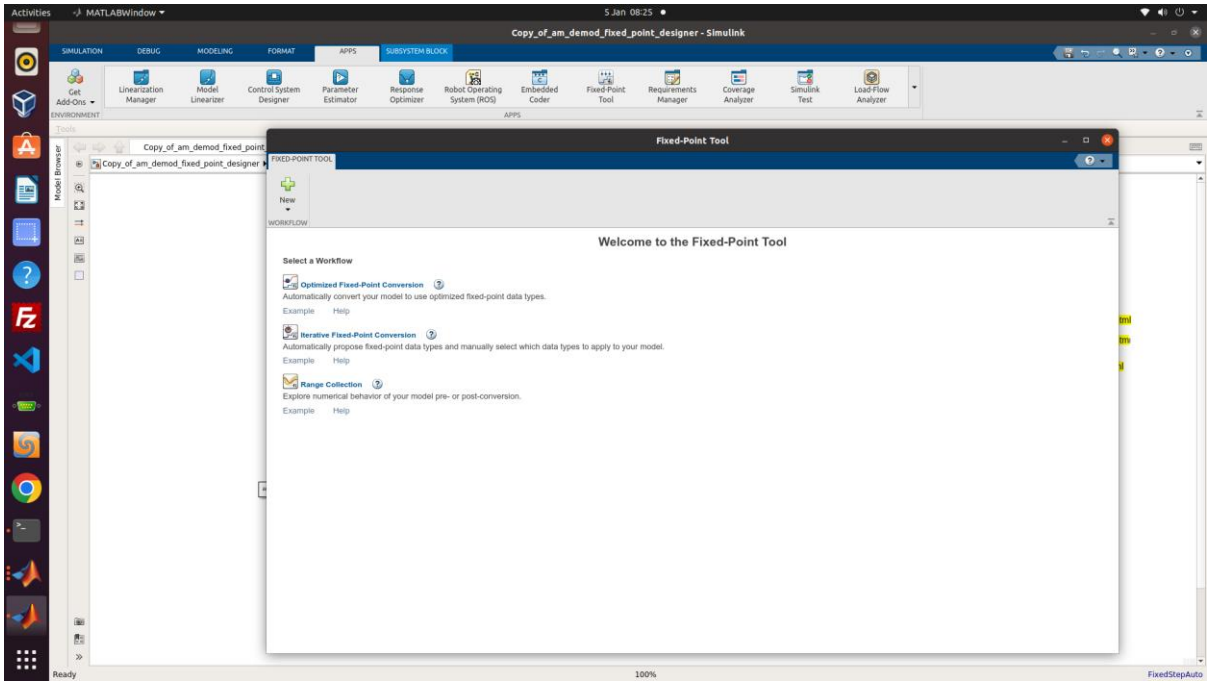
The block diagram is a minor change from the golden model, where the blocks involving reading and writing over audio files have been removed, because not compatible with fixed point designer.

Fixed point data types obtained after the model conversion are shown on each block:

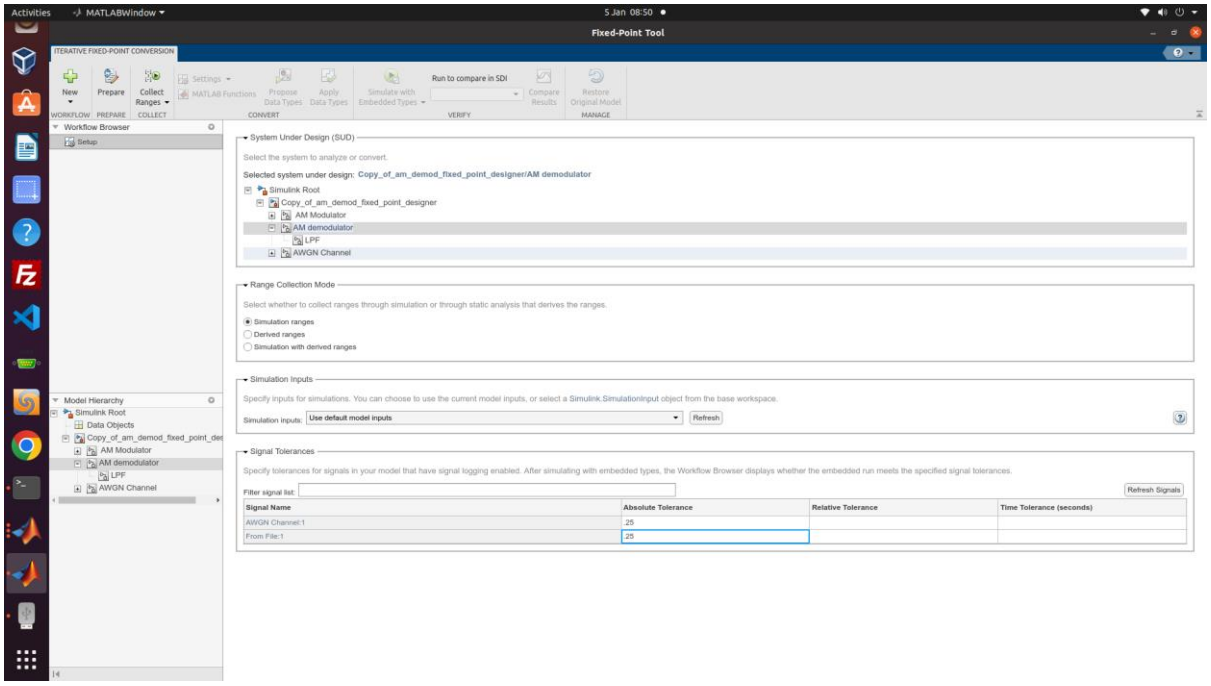




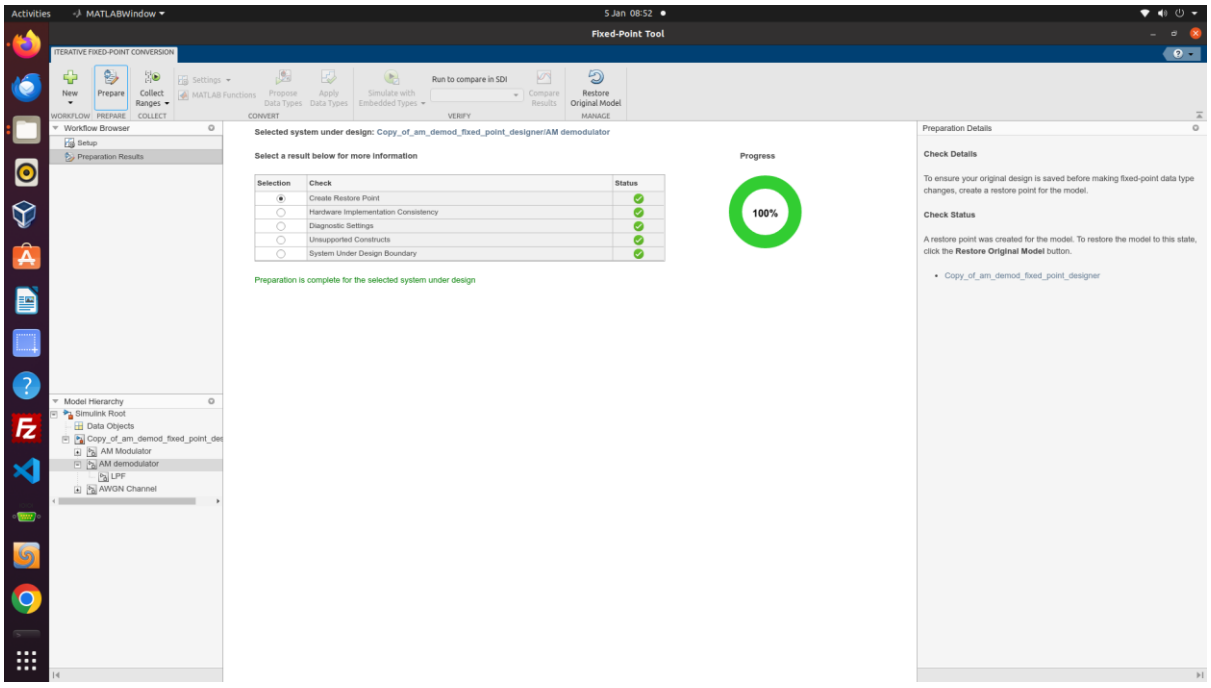
Starting from the floating point golden model block design, fixed point designer tool gets started:



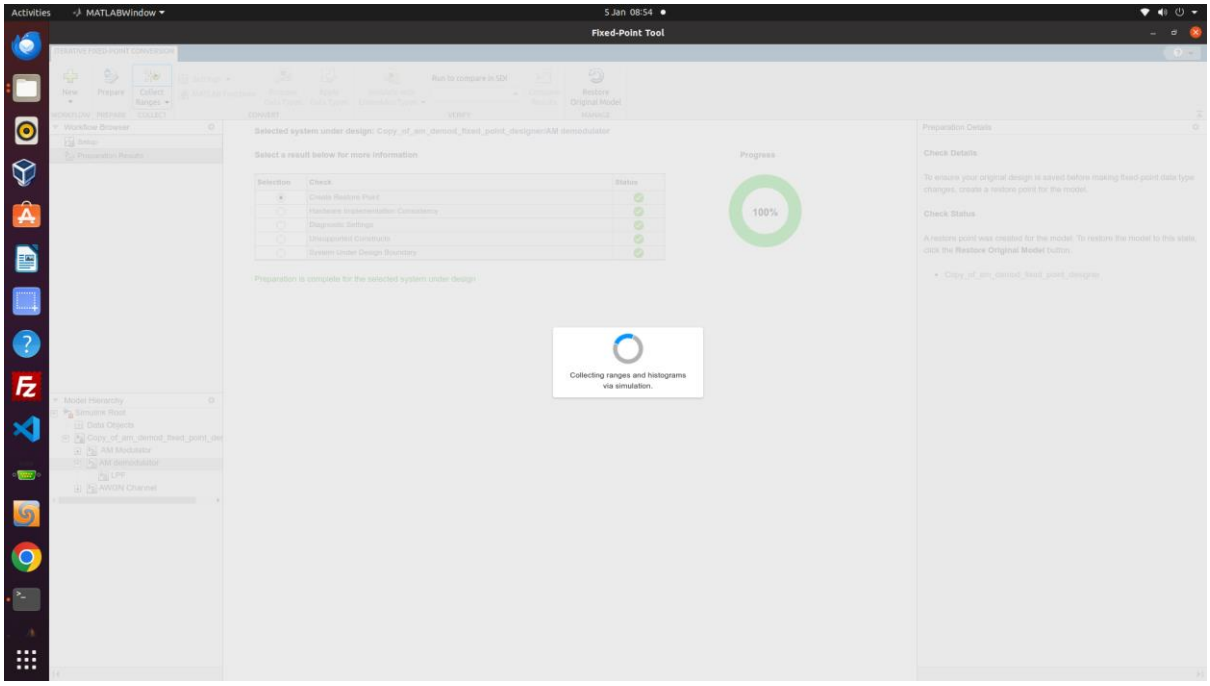
The iterative fixed point conversion option gets selected:



The prepare option gets selected, and checkings about the fixed point readiness of the model get done.



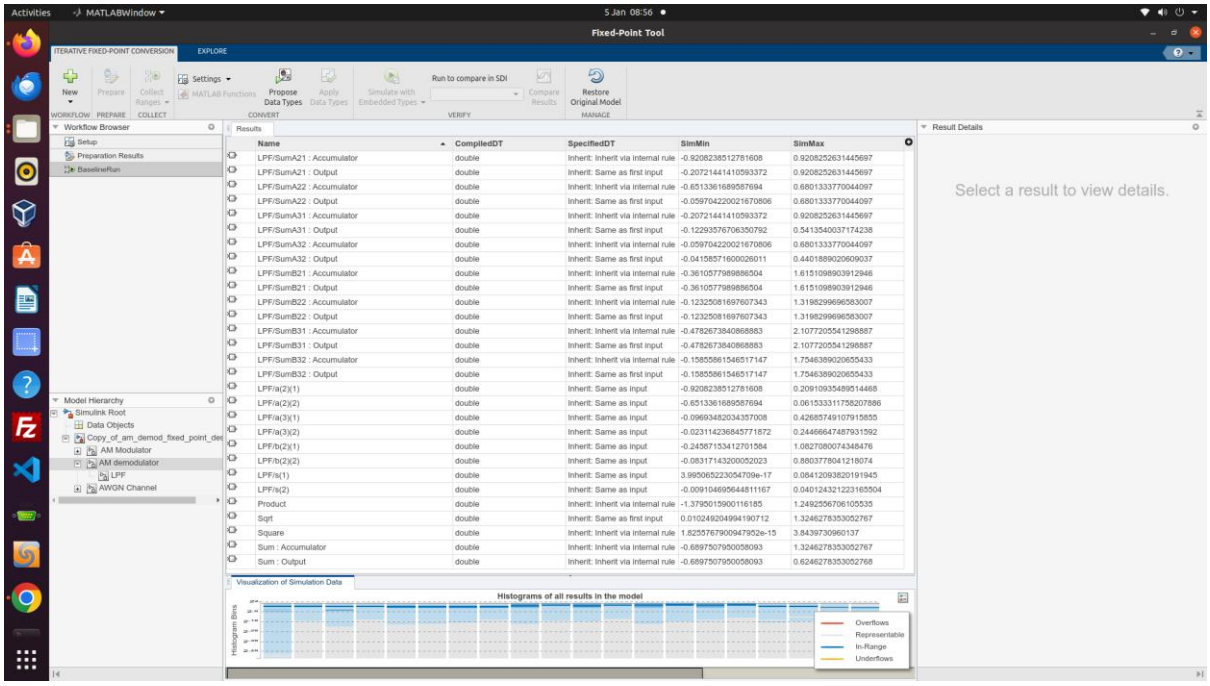
Then the “collect ranges” in double precision option gets selected:



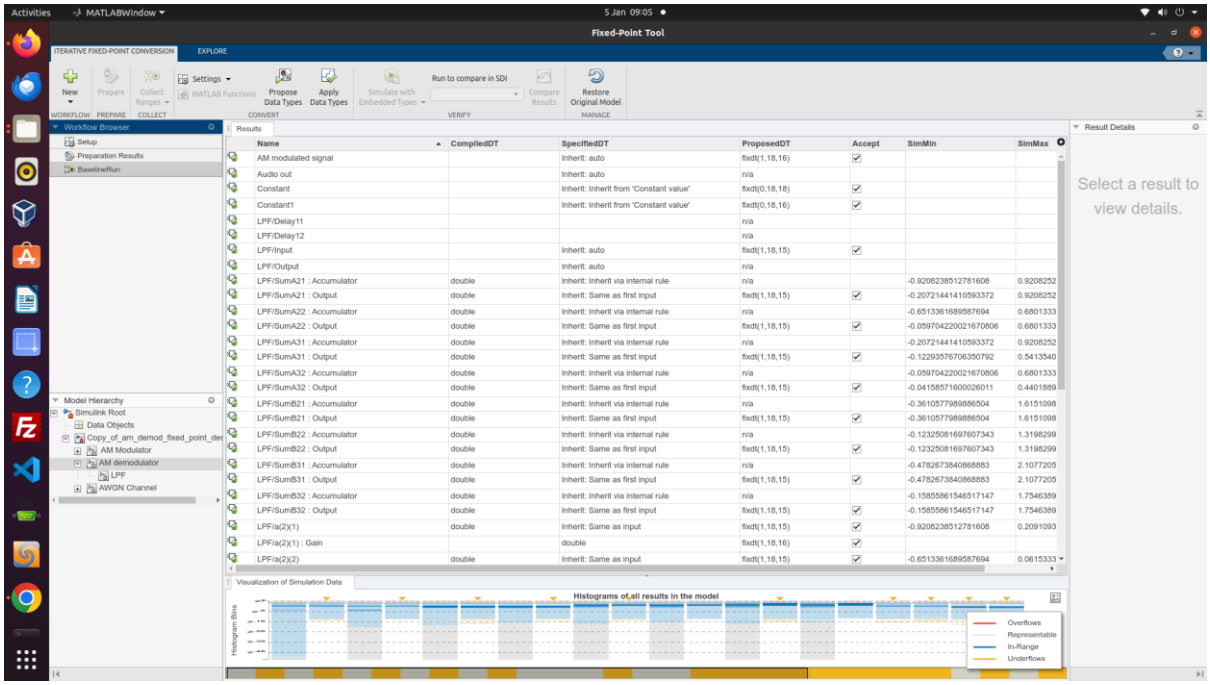
A simulation of the floating point golden reference model gets executed, and data ranges over all the nodes of the block diagram gets collected.

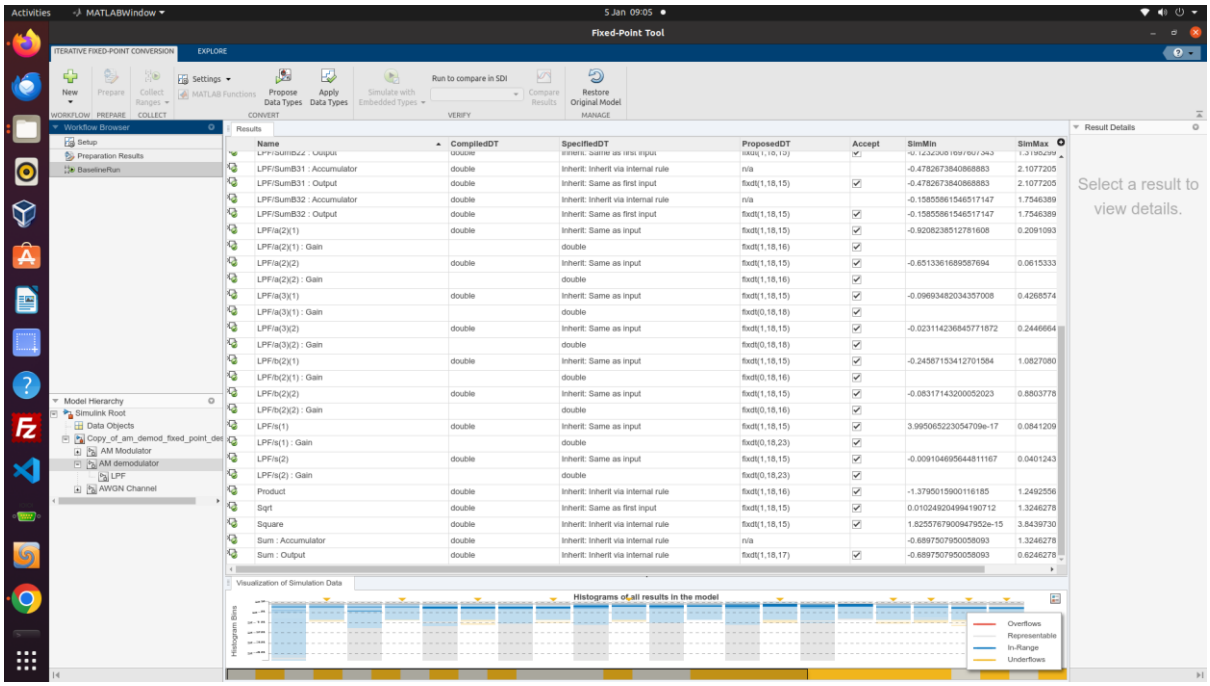
In this way, a dynamic range analysis gets performed.

This is used to find out the required fixed point data type representation to use in each point of the block diagram.

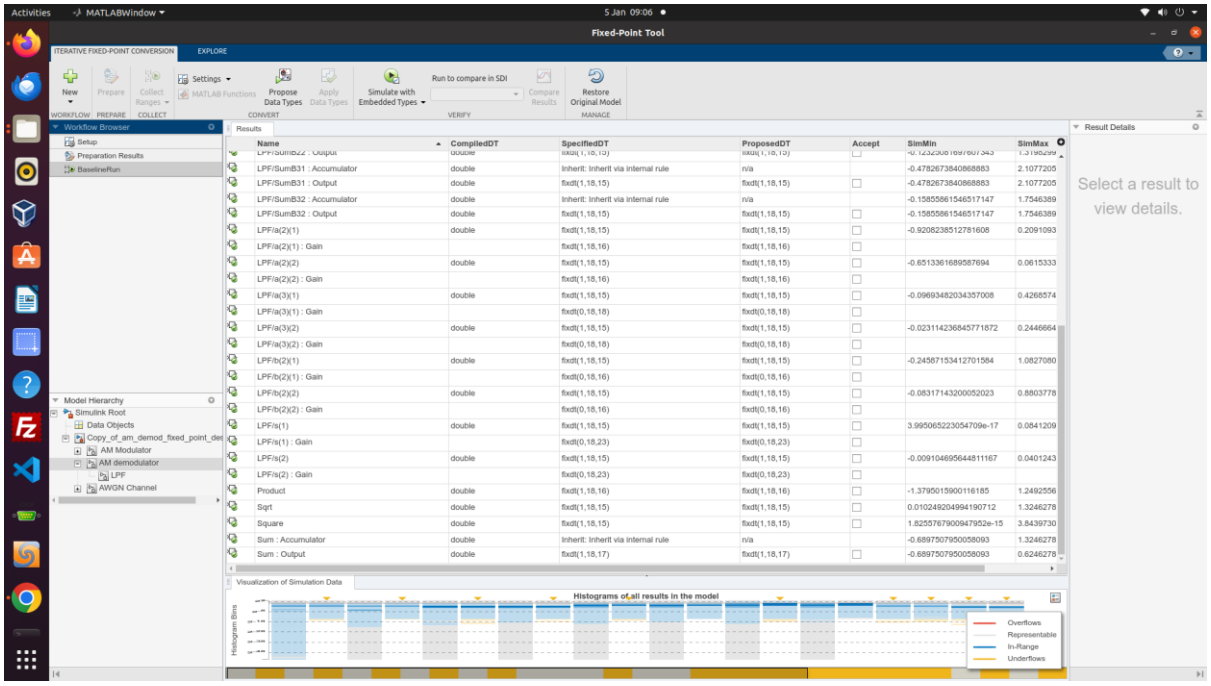


By pressing ”propose data types”, the automated fixed point data type conversion can be obtained:

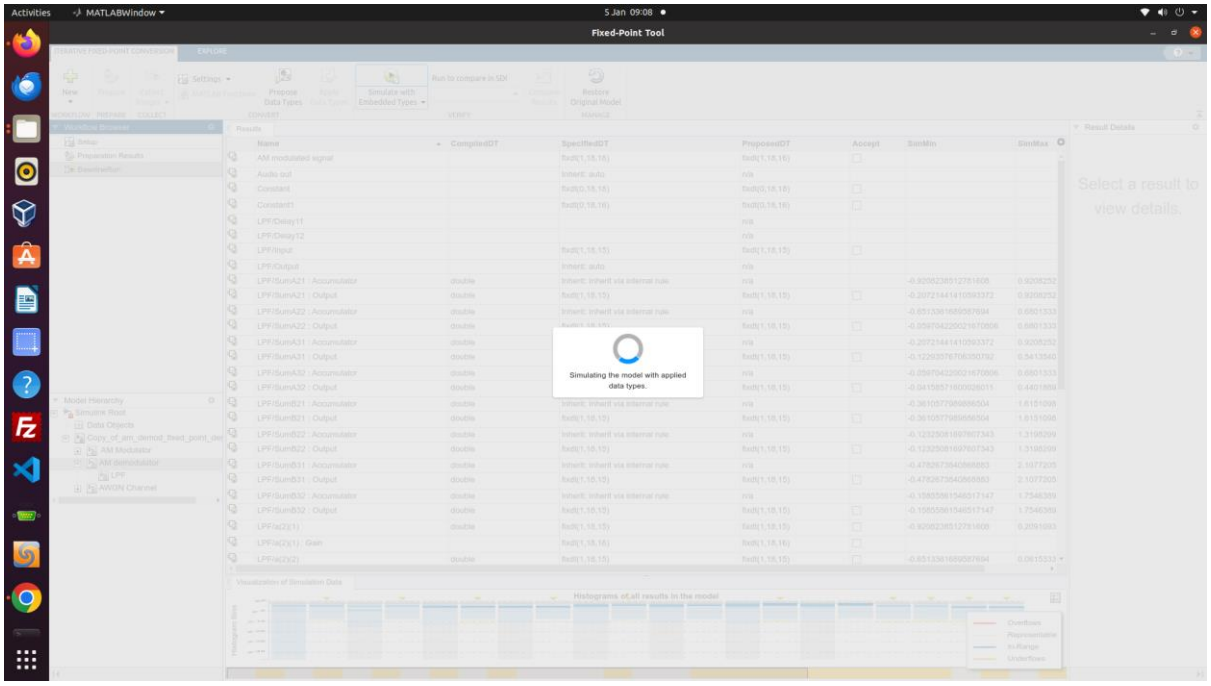




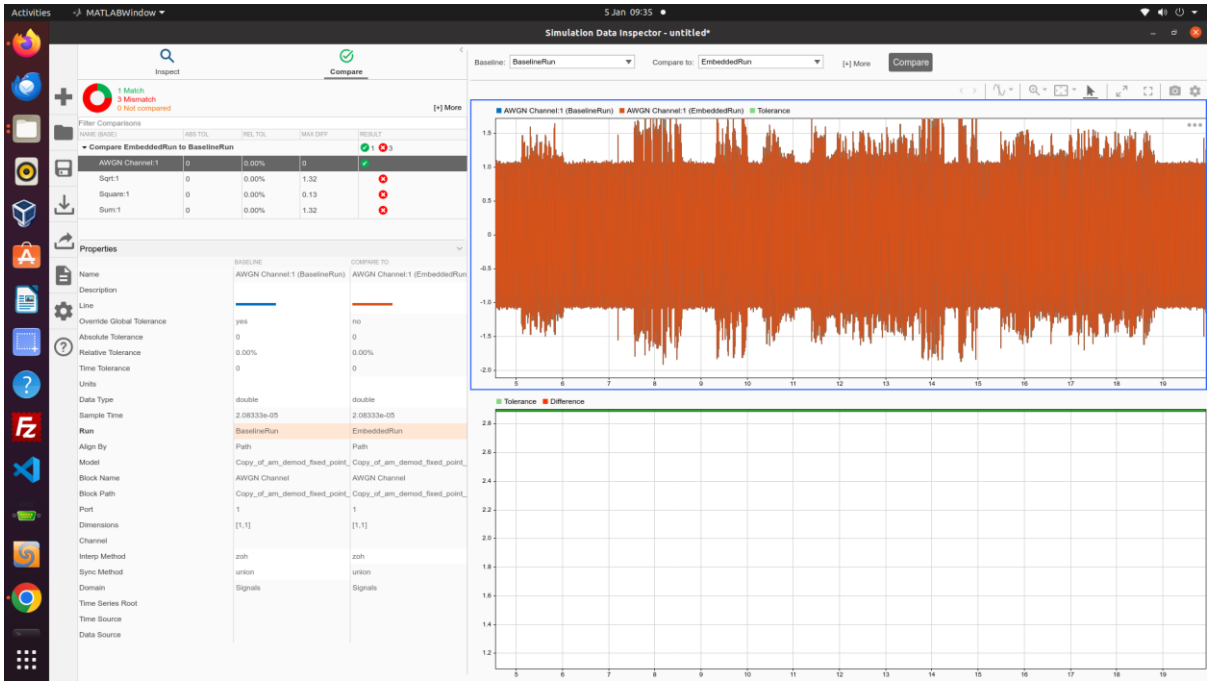
By pressing “apply data types”:



The option “simulate with embedded types” becomes available:

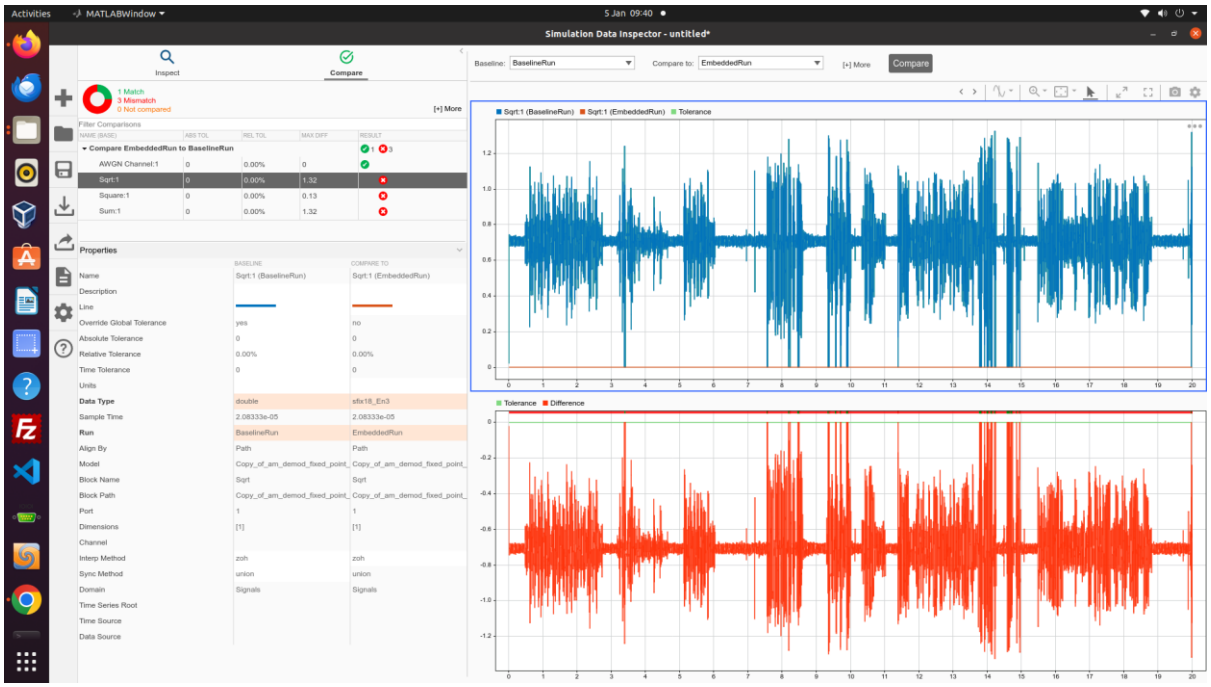


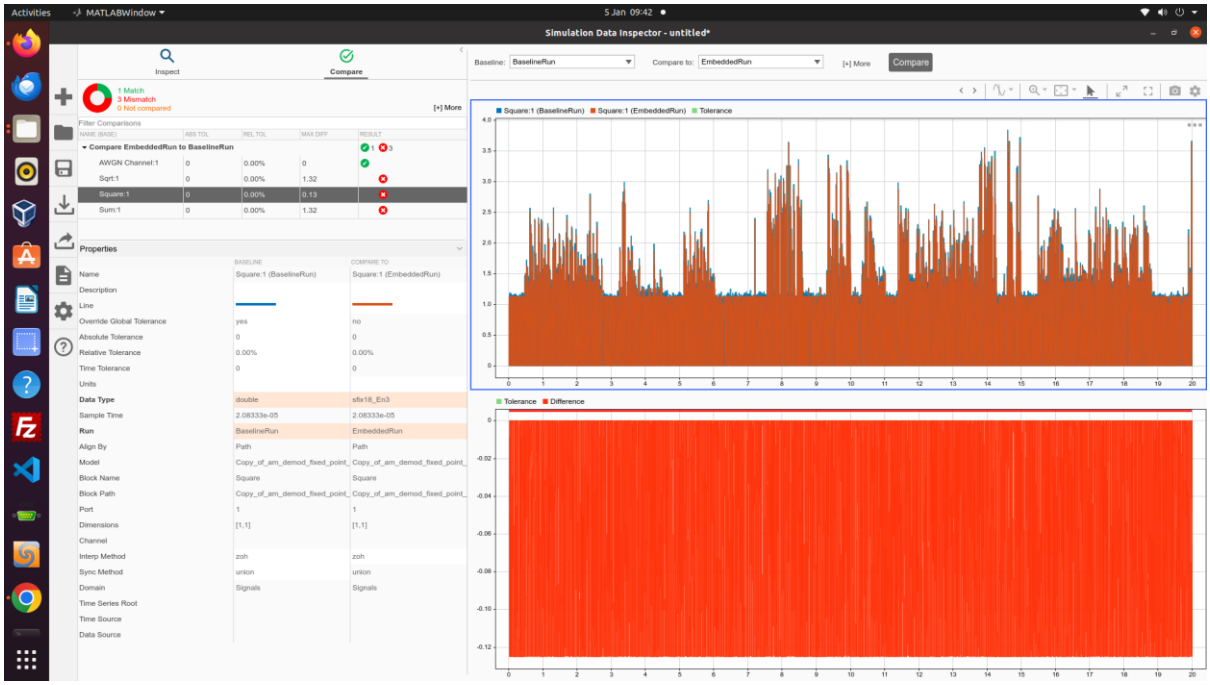
Then a report with simulation outcome of the converted model is obtained:

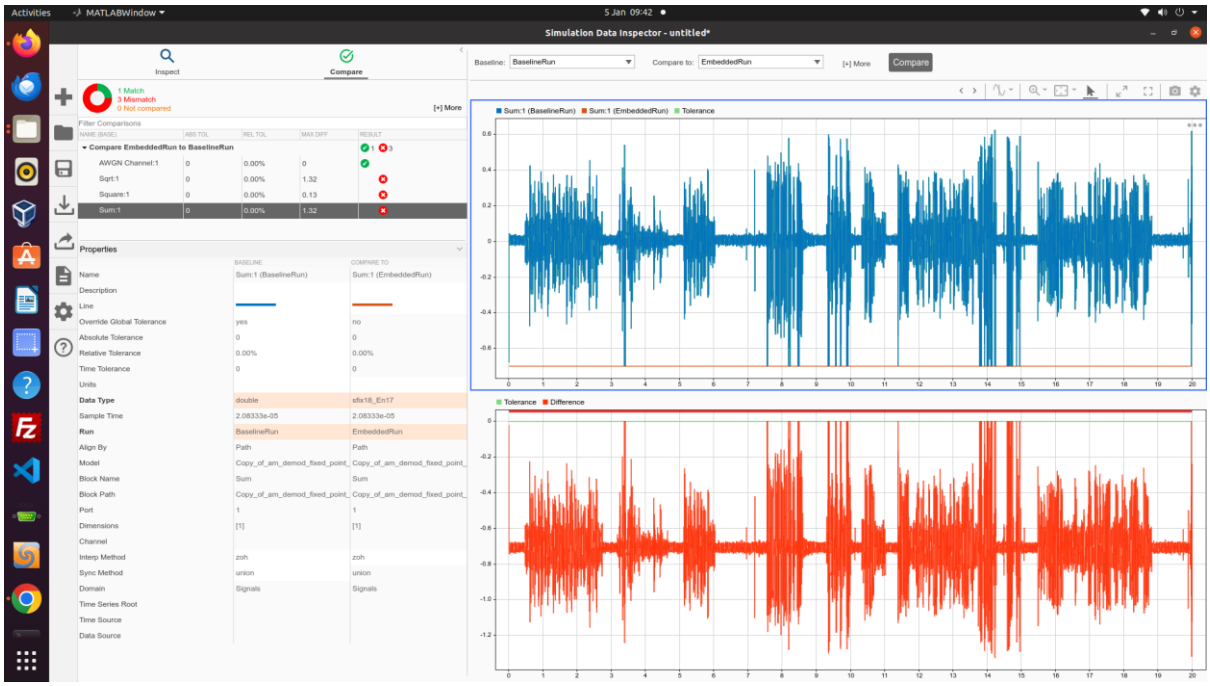


If the differences (if any) are deemed to be acceptable, the conversion finishes.

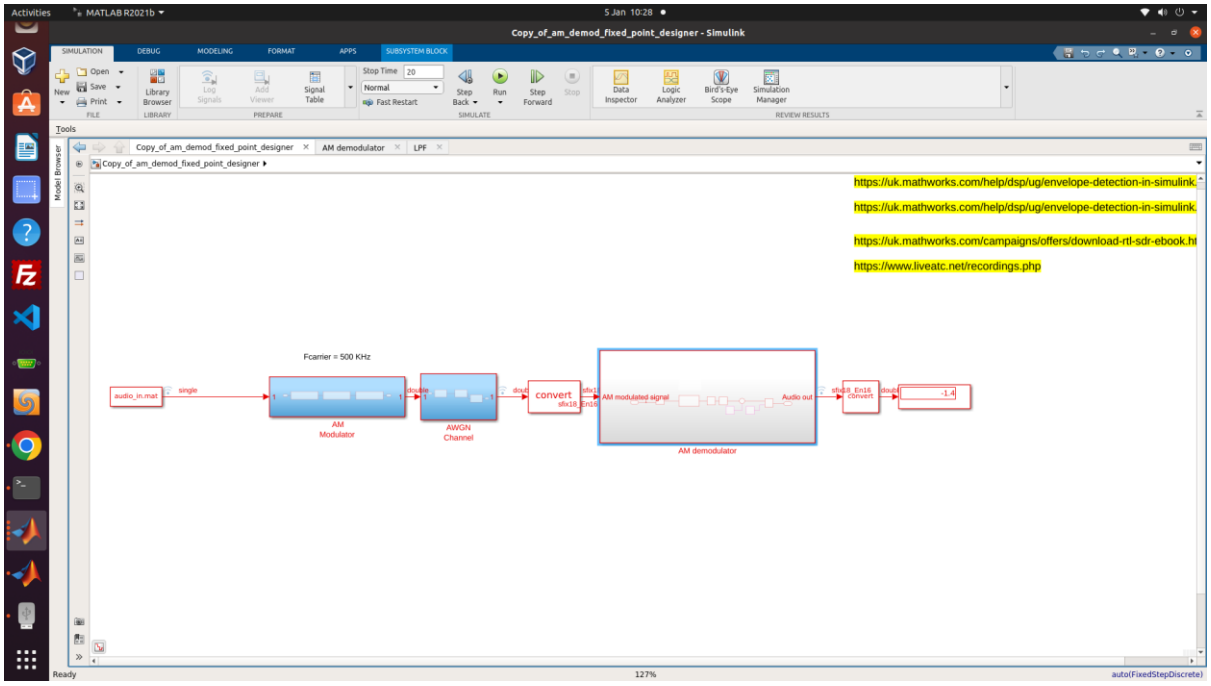
Otherwise, further iterations are possible: by tweaking precisions in nodes in need of further bits, an acceptable end result is obtained over the course of several iterations.

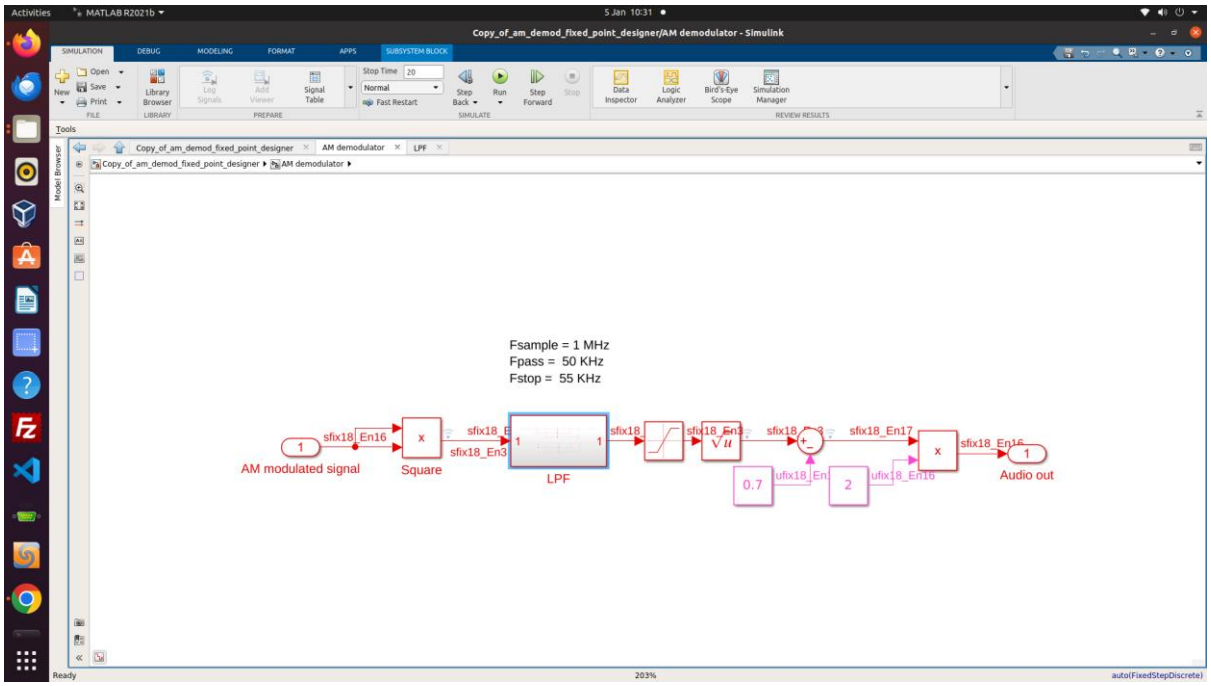


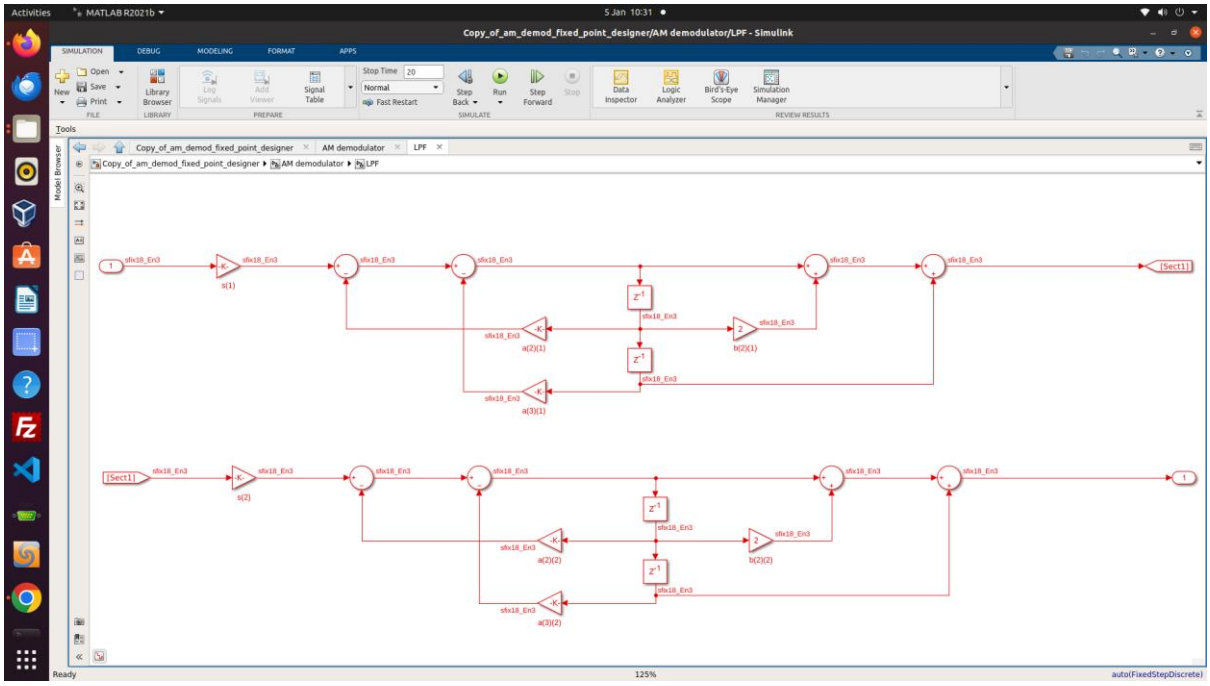




As negative values were reaching the sqrt block, a further saturation block has been added to prevent negative values reaching the block:


Activities
MATLAB R2021b
5 Jan 10:28
Copy_of_am_demod_fixed_point_designer - Simulink
SIMULATION
DEBUG
MODELING
FORMAT
APPS
SUBSYSTEM BLOCK
Tools
Copy_of_am_demod_fixed_point_designer
AM demodulator
LPF
https://uk.mathworks.com/help/dsp/ug/envelope-detection-in-simulink.
https://uk.mathworks.com/help/dsp/ug/envelope-detection-in-simulink.
https://uk.mathworks.com/campaigns/offers/download-rtl-sdr-ebook.ht
https://www.liveatc.net/recordings.php
Fcarrier = 500 KHz
audio_in.mat
single
AM Modulator
AWGN Channel
convert
AM modulated signal
Audio out
AM demodulator
-1.4
Ready
127%
auto(FixedStepDiscrete)


Activities
MATLAB R2021b
5 Jan 10:31
Copy_of_am_demod_fixed_point_designer/AM demodulator - Simulink
SIMULATION
DEBUG
MODELING
FORMAT
APPS
SUBSYSTEM BLOCK
Tools
Copy_of_am_demod_fixed_point_designer
AM demodulator
LPF
Fsample = 1 MHz
Fpass = 50 KHz
Fstop = 55 KHz
1
AM modulated signal
sfix18_En16
x
Square
sfix18_En3
LPF
sfix18
sfix18_En17
0.7
2
Audio out
Ready
203%
auto(FixedStepDiscrete)



An 18 bit word length has been used as in 7 series Xilinx devices, the DSP48e has an 18 bit wide multiplier:

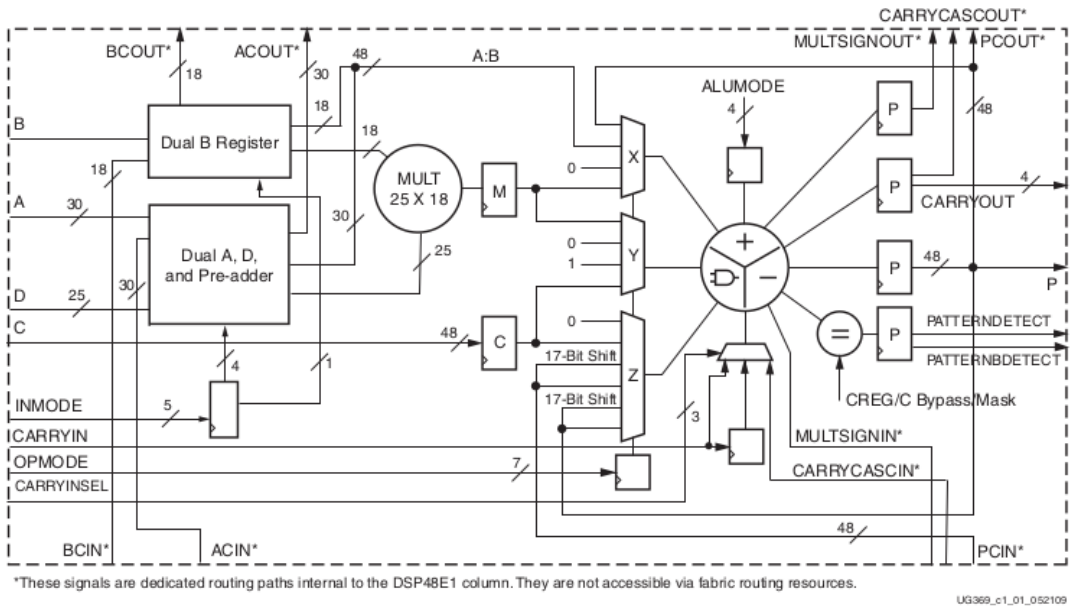


Figure 2-1: **7 Series FPGA DSP48E1 Slice**

As per UG479:

https://docs.amd.com/v/u/en-US/ug479_7Series_DSP48E1

HW/SW PARTITIONING ON MODERN SOCs AND MPSOCs

Multiprocessor System-on-Chip (MPSoC) refers to a single chip that integrates multiple processing units, which can include microprocessors, digital signal processors (DSPs), and other specialized cores. The architecture is designed to handle complex applications by distributing tasks across several processors, thereby improving performance and efficiency. MPSoCs are commonly used in embedded systems, telecommunications, automotive applications, and consumer electronics.

Hardware/Software (HW/SW) partitioning is a critical process in the design of MPSoCs. It involves dividing the system's functionality between hardware components and software processes. This partitioning aims to optimize performance metrics such as execution time, power consumption, area utilization, and overall system cost.

The primary goal of HW/SW partitioning is to determine which tasks should be executed in hardware (for speed and efficiency) and which should be implemented in software (for flexibility and ease of modification). This decision significantly impacts the system's performance characteristics.

The partitioning problem is considered complex. Factors such as inter-task data dependencies, communication overhead between hardware and software components, and resource constraints complicate the decision-making process.

Various algorithms have been developed for effective HW/SW partitioning:

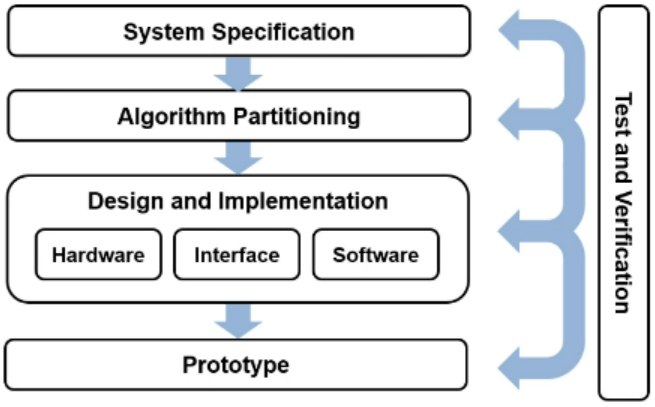
Effective HW/SW partitioning can lead to significant improvements in execution time—often reducing it by optimizing how tasks are allocated based on their computational requirements and resource availability.

https://uk.mathworks.com/videos/performing-hardware-and-software-co-design-for-xilinx-rfsocs-gen3-devices-using-matlab-and-simulink-1616608987668.html

Partitioning a design for hardware (HW) and software (SW) implementation in a Multi-Processor System on Chip (MPSoC) environment can be complex. It involves deciding which components of the system should run on programmable logic (FPGA) and which should execute on the ARM processor. MathWorks can facilitate this process through various tools and methodologies that streamline the workflow.

MathWorks

Hardware-Software Co-Design



By using Simulink, designers can create block diagrams that represent their system architecture visually. This visual representation simplifies the identification of components suitable for hardware implementation versus those better suited for software execution.

Using tools like HDL Coder™ and Embedded Coder®, MathWorks automates the generation of HDL code for FPGA and C code for ARM processors from Simulink models. This automation significantly reduces manual coding errors and accelerates the development process:

- HDL Coder: It generates synthesizable VHDL or Verilog code from Simulink models, allowing designers to implement high-performance algorithms directly onto the FPGA fabric.
- Embedded Coder: It generates optimized C code tailored for embedded systems, enabling efficient execution on ARM processors.

https://www.analog.com/en/resources/analog-dialogue/articles/using-model-based-design-sdr-1.html

https://www.analog.com/en/resources/analog-dialogue/articles/using-model-based-design-sdr-2.html

https://www.analog.com/en/resources/analog-dialogue/articles/using-model-based-design-sdr-3.html

https://www.analog.com/en/resources/analog-dialogue/articles/using-model-based-design-sdr-4.html

HDL CODER

HDL Coder is a software tool that enables high-level design for Field Programmable Gate Arrays (FPGAs), System on Chips (SoCs), and Application-Specific Integrated Circuits (ASICs). It generates portable, synthesizable hardware description language (HDL) code, specifically in Verilog®, SystemVerilog®, and VHDL® formats, from MATLAB functions, Simulink models, and Stateflow charts.

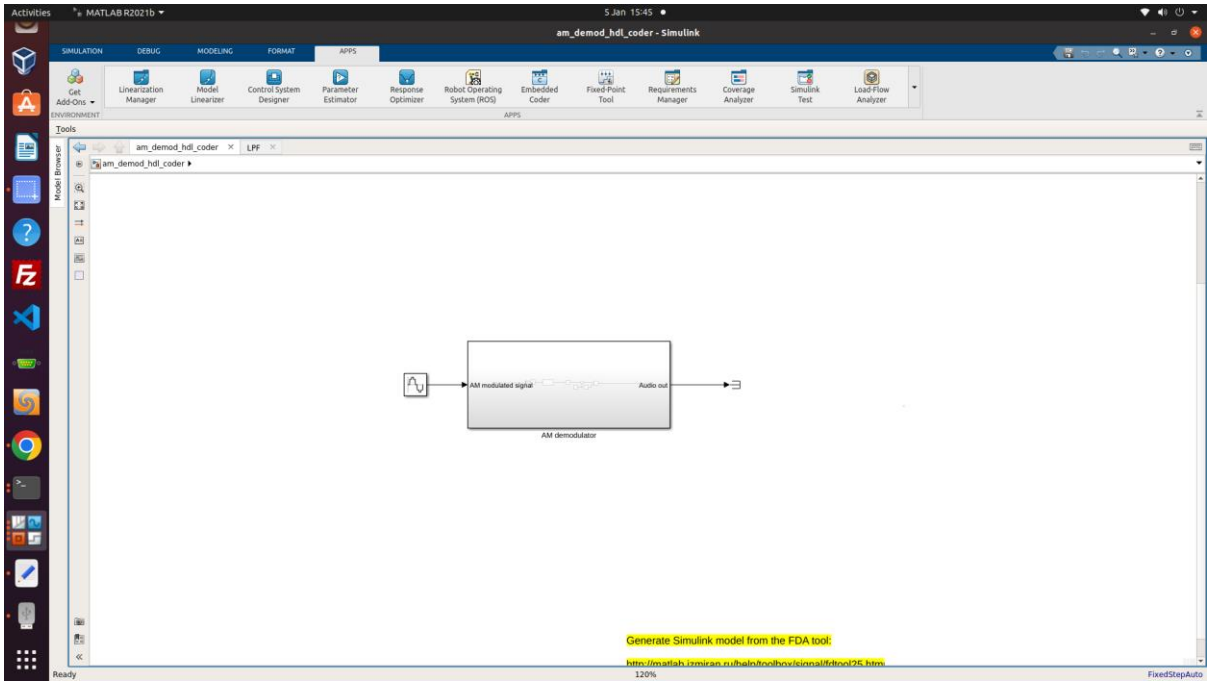
https://uk.mathworks.com/products/hdl-coder.html

This capability allows engineers to convert algorithmic designs into hardware implementations efficiently.
Key Features of HDL Coder are:

- Code Generation: HDL Coder automates the process of generating synthesizable HDL code from high-level design environments like MATLAB and Simulink. This means that users can focus on designing their algorithms without needing to manually write complex HDL code.
- Design Optimization: Users can optimize their designs for speed and area by exploring different hardware architectures and quantization options before finalizing an RTL (Register Transfer Level) implementation.
- Vendor Independence: HDL Coder generates synthesizable RTL that is optimized for FPGAs from leading vendors while also being applicable for ASIC designs. This allows for reusability of models across different platforms.
- Traceability and Verification: The tool provides traceability between Simulink models and the generated HDL code, which is crucial for verifying that the generated code meets high-integrity application standards like DO-254. This ensures that the design adheres to necessary regulatory requirements.
- Real-Time Simulation and Testing: HDL Coder supports real-time simulation using tools like Simulink Real-Time, enabling users to test their designs in a simulated environment before deploying them on actual hardware.

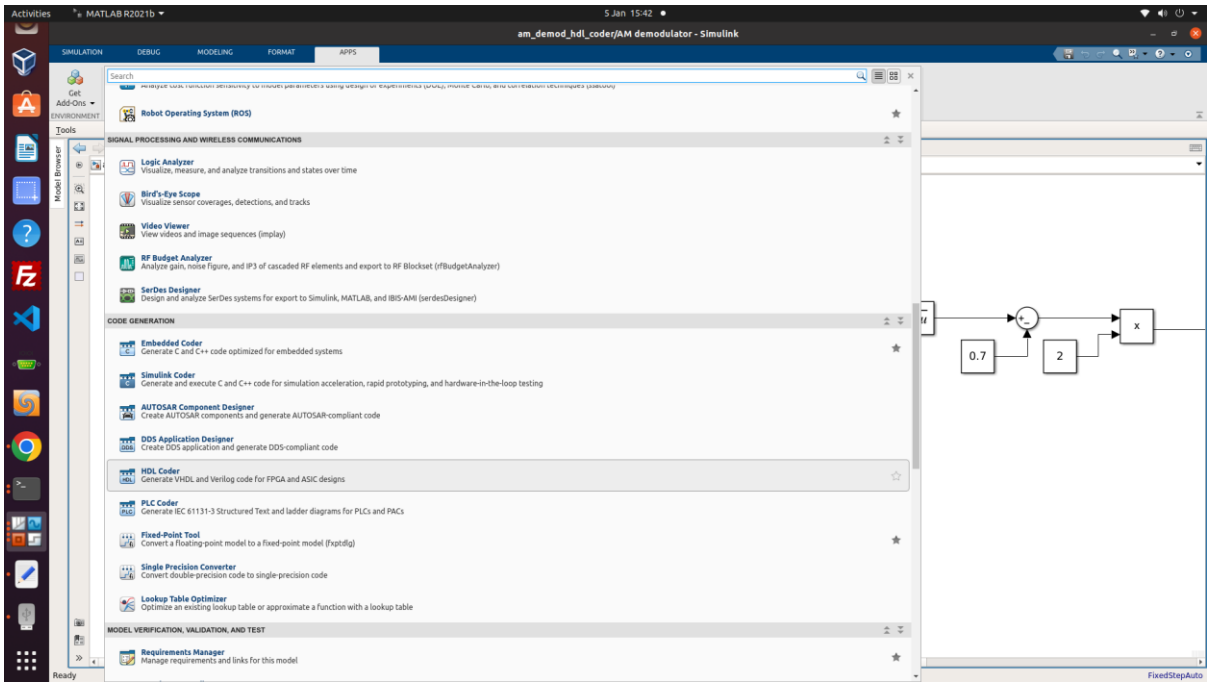
https://uk.mathworks.com/videos/software-defined-radio-using-matlab-and-simulink-1601496230597.html

The starting model used as an input to HDL coder is essentially the same used in the golden reference model:

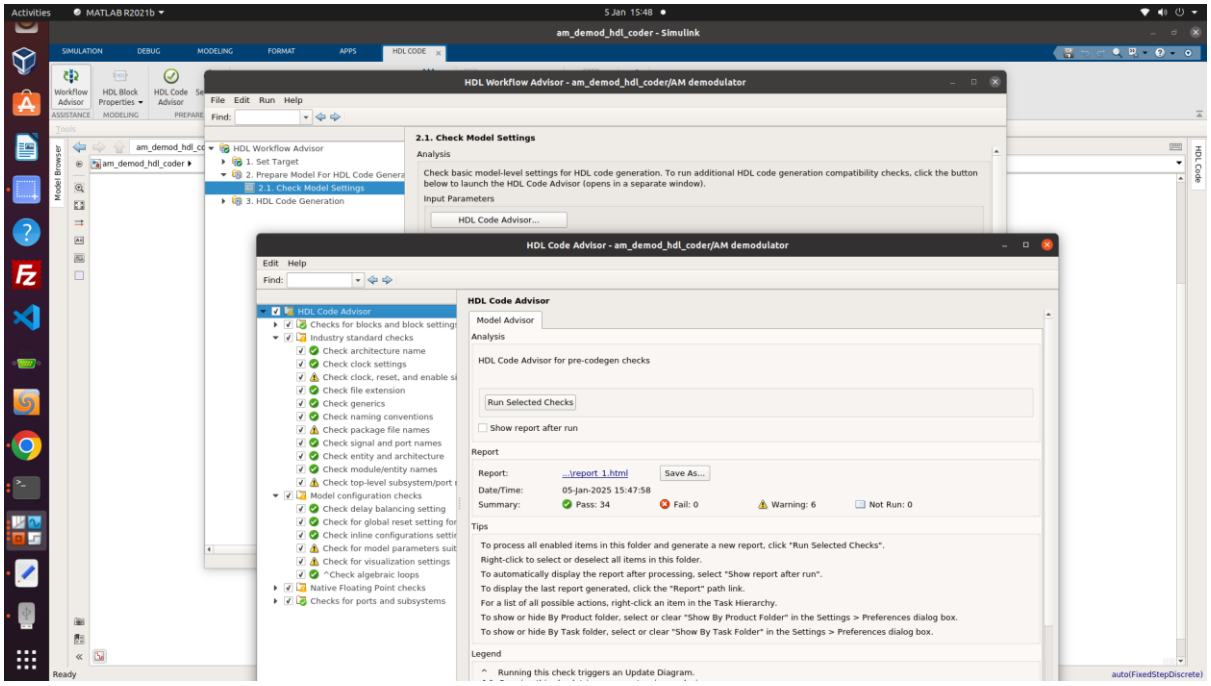


Also in this case audio file blocks have been removed, as being incompatible with the tool.

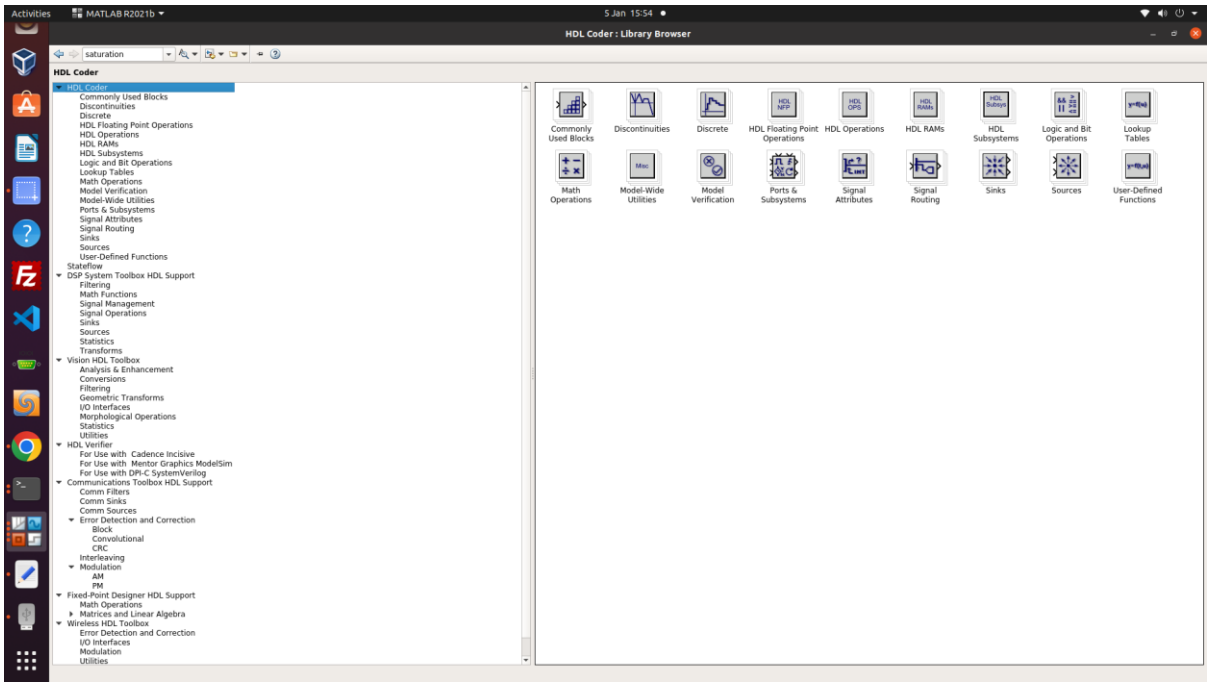
HDL coder gets started as follows:



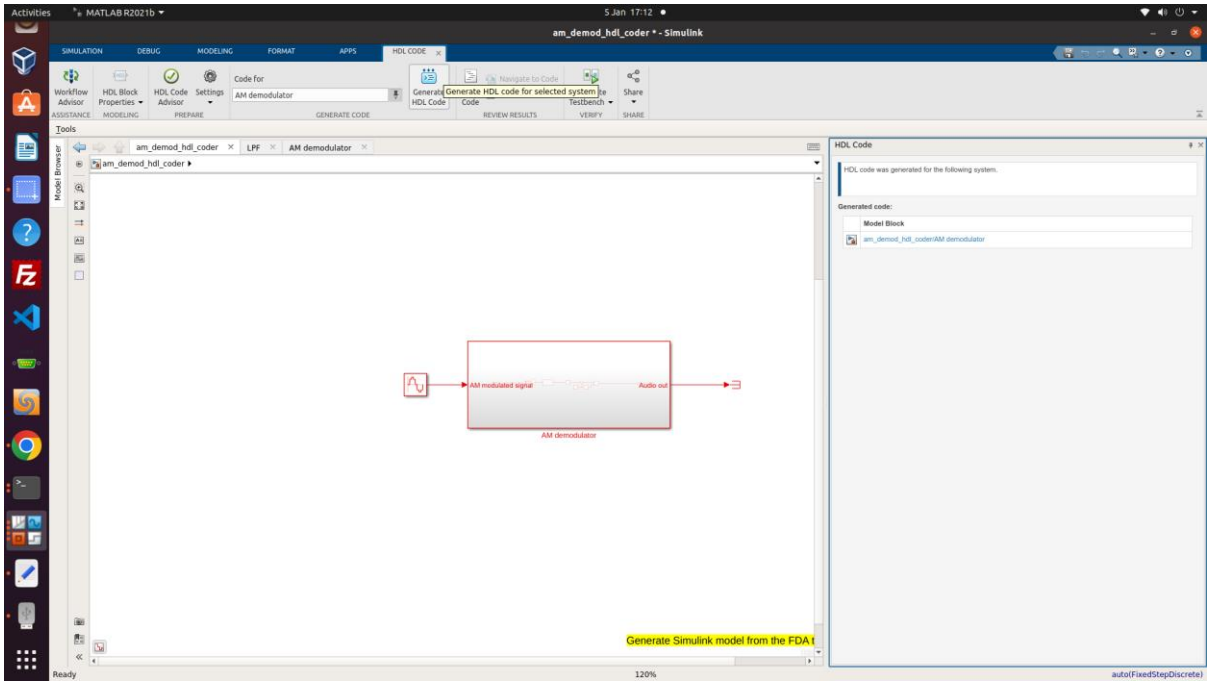
Preliminary checkings on the model can be run via “workflow advisor”:



The number of blocks suitable for automatic HDL generation, is quite big, but not all of Simulink blocks are usable though:

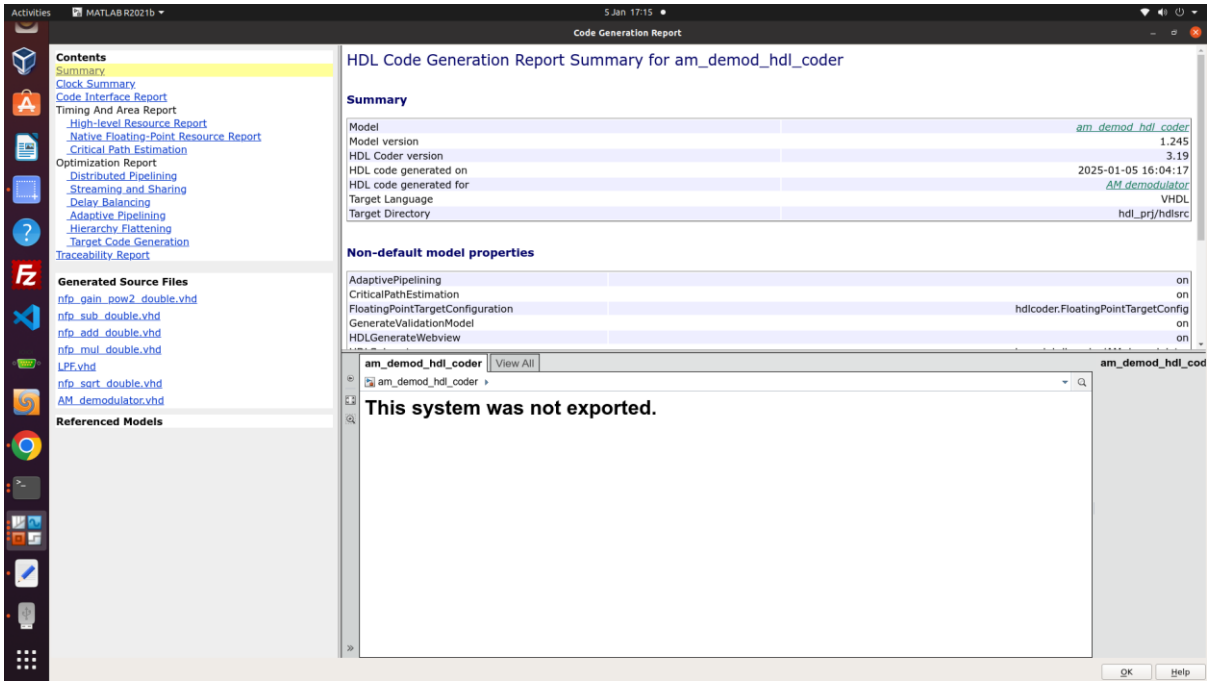


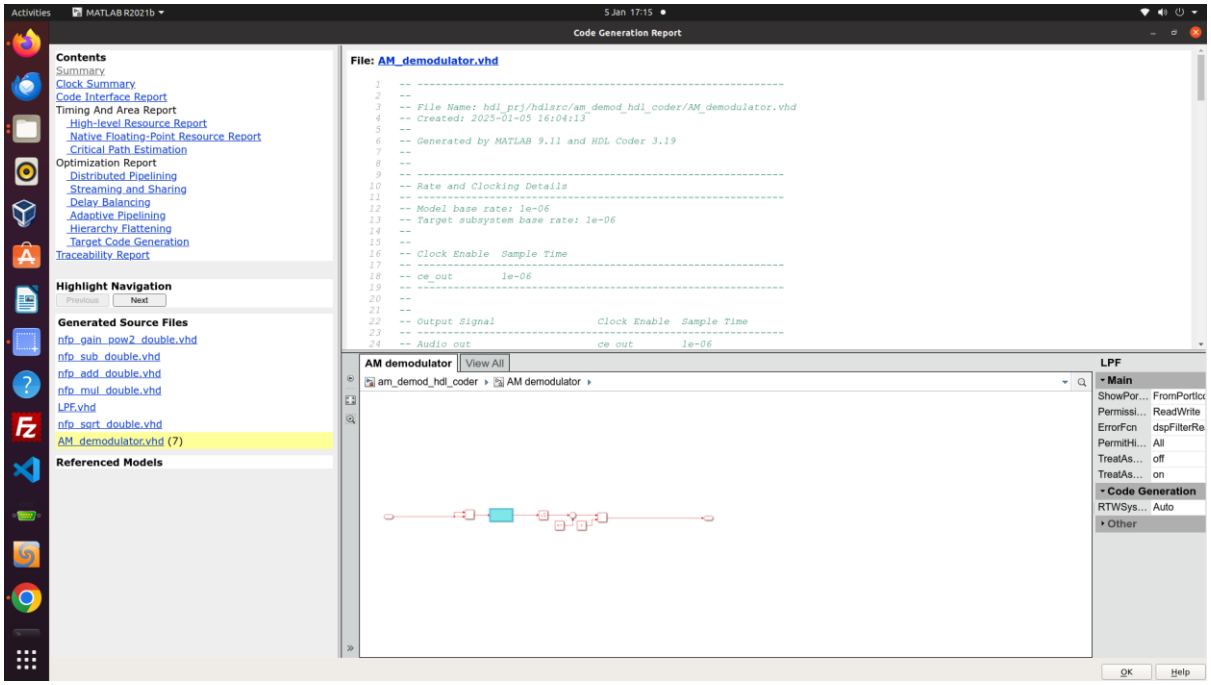
By pressing the generate HDL code:

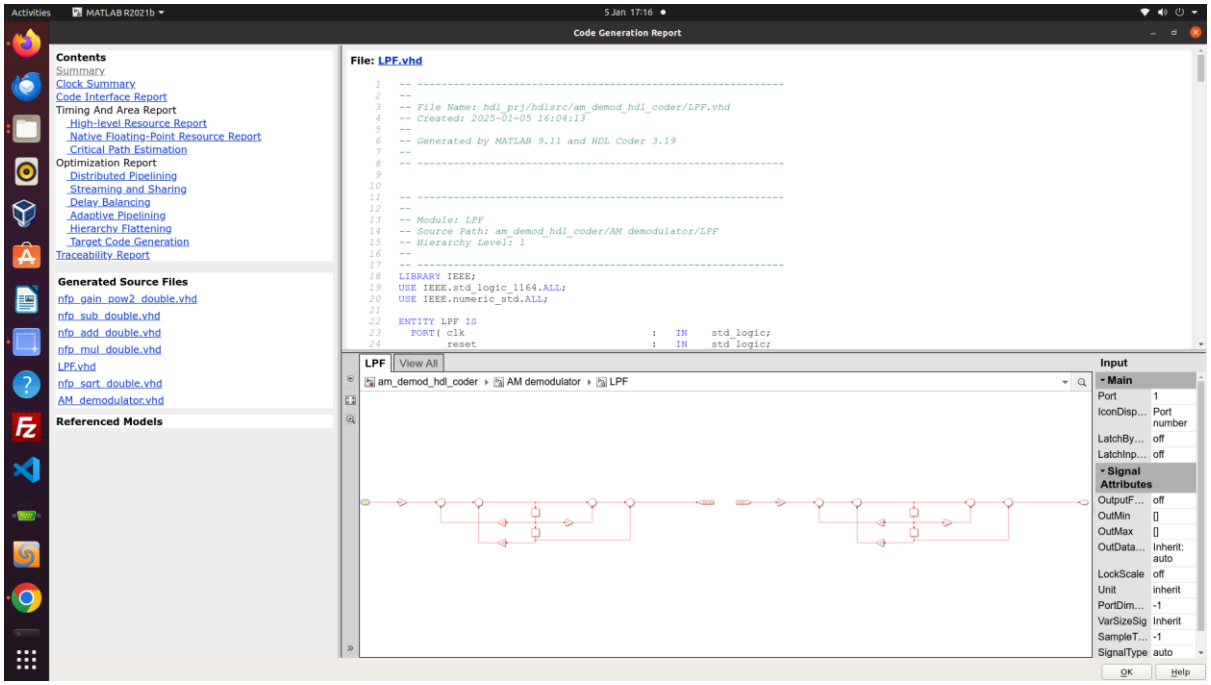


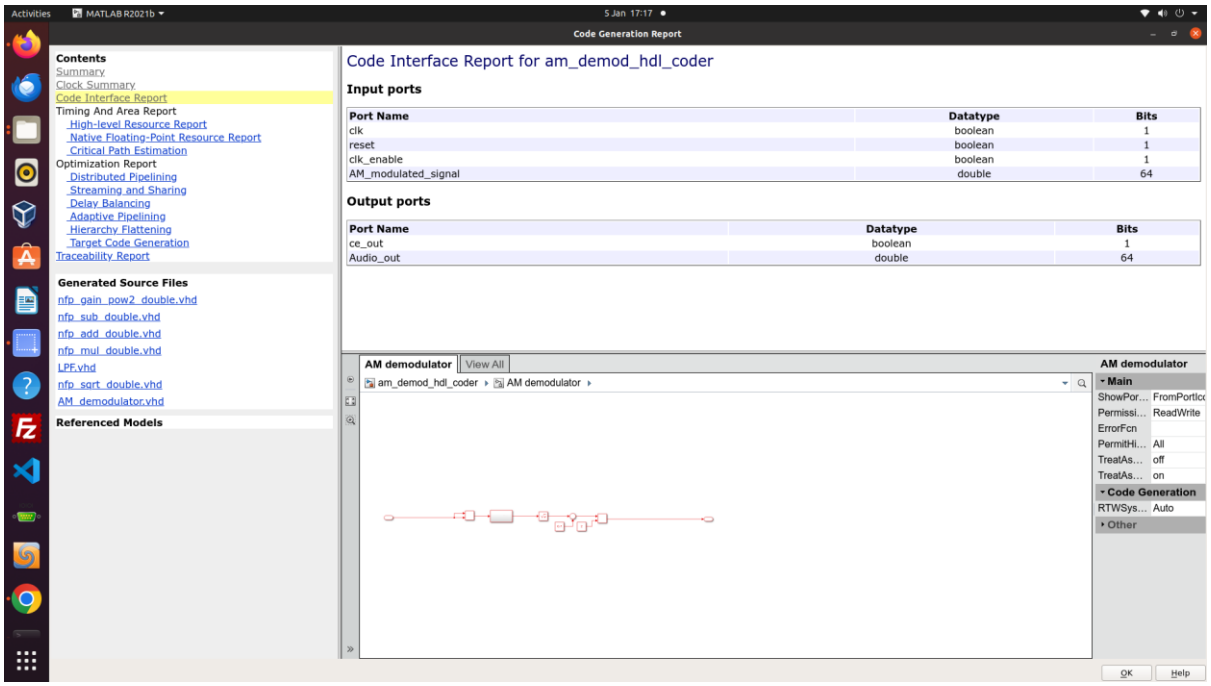
The HDL code for the block diagram in the HDL language and architecture of choice gets generated.

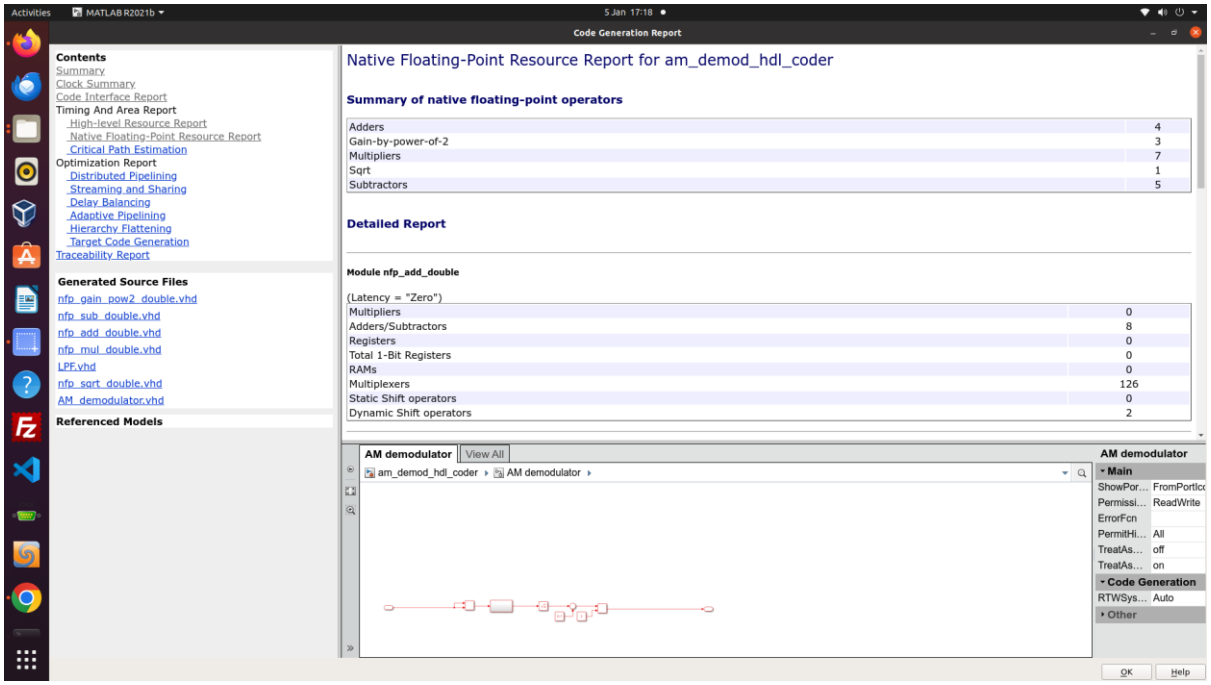
An in depth report is created as well: it details resources utilization, performances, traces back the HDL code to Simulink blocks too.

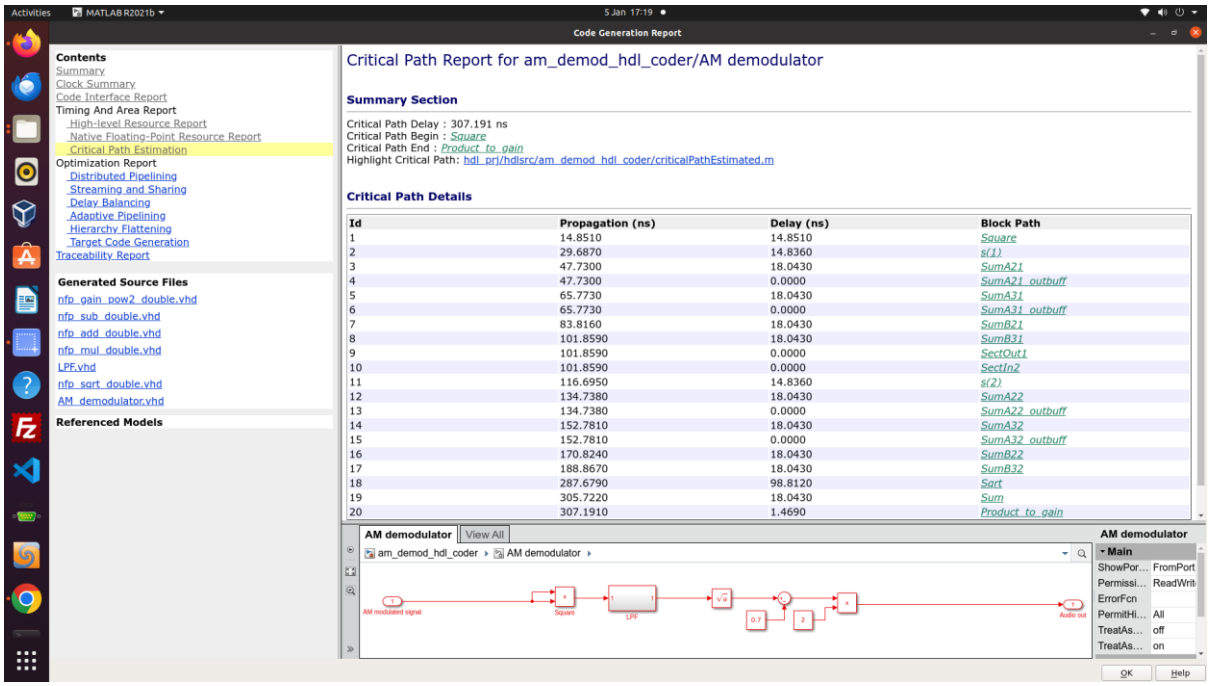


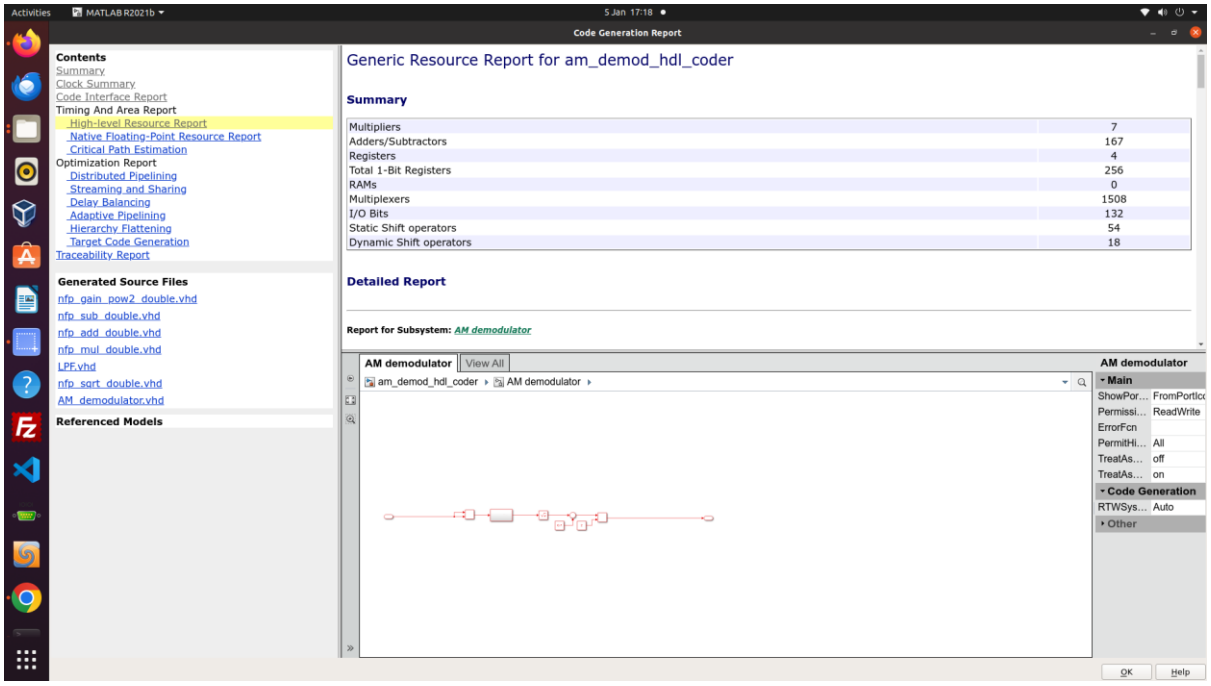

Activities
MATLAB R2021b
5 Jan 17:15
Code Generation Report
Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Native Floating-Point Resource Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Delay Balancing
Adaptive Pipelining
Hierarchy Flattening
Target Code Generation
Traceability Report
Highlight Navigation
Previous
Next
Generated Source Files
nfp_gain_pow2_double.vhd
nfp_sub_double.vhd
nfp_add_double.vhd
nfp_mul_double.vhd
LPF.vhd
nfp_sqrt_double.vhd
AM_demodulator.vhd (7)
Referenced Models
File: AM_demodulator.vhd
-- ---
-- File Name: hdl_prj/hdlsrc/am_demod_hdl_coder/AM_demodulator.vhd
-- Created: 2025-01-05 16:04:13
-- Generated by MATLAB 9.11 and HDL Coder 3.19
-- Rate and Clocking Details
-- Model base rate: 1e-06
-- Target subsystem base rate: 1e-06
-- Clock Enable Sample Time
-- ce_out 1e-06
-- Output Signal Clock Enable Sample Time
-- Audio out ce_out 1e-06
AM demodulator
View All
am_demod_hdl_coder
AM demodulator
LPF
Main
ShowPor... FromPortIc
Permissi... ReadWrite
ErrorFcn dspFilterRe
PermitHi... All
TreatAs... off
TreatAs... on
Code Generation
RTWSys... Auto
Other
OK
Help

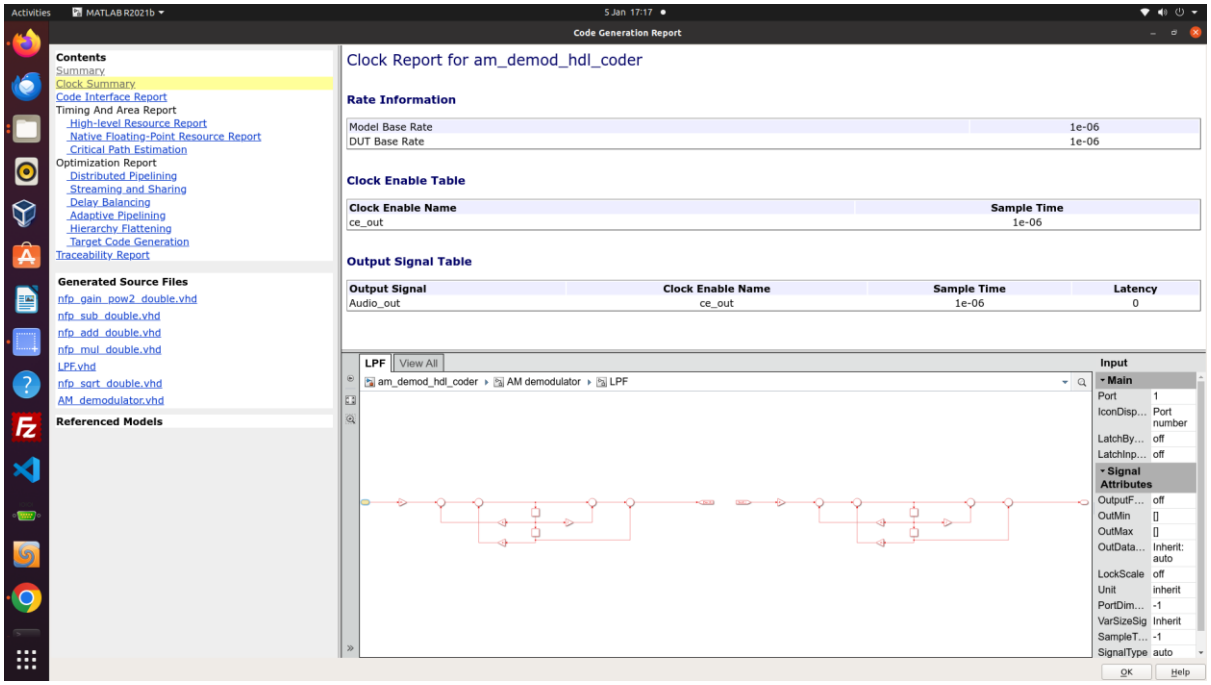

Activities
MATLAB R2021b
5 Jan 17:16
Code Generation Report
Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Native Floating-Point Resource Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Delay Balancing
Adaptive Pipelining
Hierarchy Flattening
Target Code Generation
Traceability Report
Generated Source Files
nfp_gain_pow2_double.vhd
nfp_sub_double.vhd
nfp_add_double.vhd
nfp_mul_double.vhd
LPF.vhd
nfp_sqrt_double.vhd
AM_demodulator.vhd
Referenced Models
File: LPF.vhd
-- File Name: hdl_prj/hdlsrc/am_demod_hdl_coder/LPF.vhd
-- Created: 2025-01-05 16:04:13
-- Generated by MATLAB 9.11 and HDL Coder 3.19
-- Module: LPF
-- Source Path: am_demod_hdl_coder/AM demodulator/LPF
-- Hierarchy Level: 1
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
ENTITY LPF IS
PORT(clk : IN std_logic;
reset : IN std_logic;
LPF
View All
am_demod_hdl_coder
AM demodulator
LPF
Input
Main
Port 1
IconDisp... Port number
LatchBy... off
LatchInp... off
Signal Attributes
OutputF... off
OutMin []
OutMax []
OutData... Inherit: auto
LockScale off
Unit inherit
PortDim... -1
VarSizeSig Inherit
SampleT... -1
SignalType auto
OK
Help

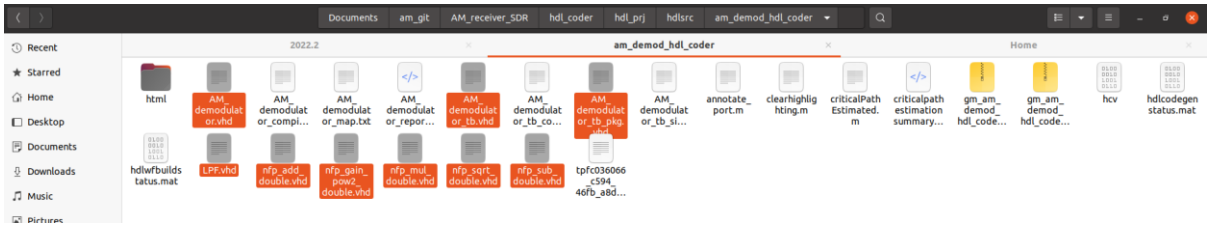

Activities
MATLAB R2021b
5 Jan 17:17
Code Generation Report
Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Native Floating-Point Resource Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Delay Balancing
Adaptive Pipelining
Hierarchy Flattening
Target Code Generation
Traceability Report
Generated Source Files
nfp_gain_pow2_double.vhd
nfp_sub_double.vhd
nfp_add_double.vhd
nfp_mul_double.vhd
LPF.vhd
nfp_sqrt_double.vhd
AM_demodulator.vhd
Referenced Models
Code Interface Report for am_demod_hdl_coder
Input ports
Port Name Datatype Bits
clk boolean 1
reset boolean 1
clk_enable boolean 1
AM_modulated_signal double 64
Output ports
Port Name Datatype Bits
ce_out boolean 1
Audio_out double 64
AM demodulator
View All
am_demod_hdl_coder
AM demodulator
AM demodulator
Main
ShowPor... FromPortIc
Permissi... ReadWrite
ErrorFcn
PermitHi... All
TreatAs... off
TreatAs... on
Code Generation
RTWSys... Auto
Other
OK
Help


Activities
MATLAB R2021b
5 Jan 17:18
Code Generation Report
Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Native Floating-Point Resource Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Delay Balancing
Adaptive Pipelining
Hierarchy Flattening
Target Code Generation
Traceability Report
Generated Source Files
nfp_gain_pow2_double.vhd
nfp_sub_double.vhd
nfp_add_double.vhd
nfp_mul_double.vhd
LPF.vhd
nfp_sqrt_double.vhd
AM_demodulator.vhd
Referenced Models
Native Floating-Point Resource Report for am_demod_hdl_coder
Summary of native floating-point operators
Adders 4
Gain-by-power-of-2 3
Multipliers 7
Sqrt 1
Subtractors 5
Detailed Report
Module nfp_add_double
(Latency = "Zero")
Multipliers 0
Adders/Subtractors 8
Registers 0
Total 1-Bit Registers 0
RAMs 0
Multiplexers 126
Static Shift operators 0
Dynamic Shift operators 2
AM demodulator
View All
am_demod_hdl_coder
AM demodulator
Main
Code Generation
Other
OK
Help


Activities
MATLAB R2021b
5 Jan 17:19
Code Generation Report
Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Native Floating-Point Resource Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Delay Balancing
Adaptive Pipelining
Hierarchy Flattening
Target Code Generation
Traceability Report
Generated Source Files
nfp_gain_pow2_double.vhd
nfp_sub_double.vhd
nfp_add_double.vhd
nfp_mul_double.vhd
LPF.vhd
nfp_sqrt_double.vhd
AM_demodulator.vhd
Referenced Models
Critical Path Report for am_demod_hdl_coder/AM demodulator
Summary Section
Critical Path Delay : 307.191 ns
Critical Path Begin : Square
Critical Path End : Product_to_gain
Highlight Critical Path: hdl_prj/hdlsrc/am_demod_hdl_coder/criticalPathEstimated.m
Critical Path Details
Id | Propagation (ns) | Delay (ns) | Block Path
1 | 14.8510 | 14.8510 | Square
2 | 29.6870 | 14.8360 | s(1)
3 | 47.7300 | 18.0430 | SumA21
4 | 47.7300 | 0.0000 | SumA21_outbuff
5 | 65.7730 | 18.0430 | SumA31
6 | 65.7730 | 0.0000 | SumA31_outbuff
7 | 83.8160 | 18.0430 | SumB21
8 | 101.8590 | 18.0430 | SumB31
9 | 101.8590 | 0.0000 | SectOut1
10 | 101.8590 | 0.0000 | SectIn2
11 | 116.6950 | 14.8360 | s(2)
12 | 134.7380 | 18.0430 | SumA22
13 | 134.7380 | 0.0000 | SumA22_outbuff
14 | 152.7810 | 18.0430 | SumA32
15 | 152.7810 | 0.0000 | SumA32_outbuff
16 | 170.8240 | 18.0430 | SumB22
17 | 188.8670 | 18.0430 | SumB32
18 | 287.6790 | 98.8120 | Sqrt
19 | 305.7220 | 18.0430 | Sum
20 | 307.1910 | 1.4690 | Product_to_gain
AM demodulator
View All
am_demod_hdl_coder
AM demodulator
Main
AM modulated signal
Square
LPF
0.7
2
Audio out
OK
Help


Activities
MATLAB R2021b
5 Jan 17:18
Code Generation Report
Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Native Floating-Point Resource Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Delay Balancing
Adaptive Pipelining
Hierarchy Flattening
Target Code Generation
Traceability Report
Generated Source Files
nfp_gain_pow2_double.vhd
nfp_sub_double.vhd
nfp_add_double.vhd
nfp_mul_double.vhd
LPF.vhd
nfp_sqrt_double.vhd
AM_demodulator.vhd
Referenced Models
Generic Resource Report for am_demod_hdl_coder
Summary
Multipliers 7
Adders/Subtractors 167
Registers 4
Total 1-Bit Registers 256
RAMs 0
Multiplexers 1508
I/O Bits 132
Static Shift operators 54
Dynamic Shift operators 18
Detailed Report
Report for Subsystem: AM demodulator
AM demodulator
View All
am_demod_hdl_coder
AM demodulator
Main
Code Generation
Other
OK
Help


Activities
MATLAB R2021b
5 Jan 17:17
Code Generation Report
Contents
Summary
Clock Summary
Code Interface Report
Timing And Area Report
High-level Resource Report
Native Floating-Point Resource Report
Critical Path Estimation
Optimization Report
Distributed Pipelining
Streaming and Sharing
Delay Balancing
Adaptive Pipelining
Hierarchy Flattening
Target Code Generation
Traceability Report
Generated Source Files
nfp_gain_pow2_double.vhd
nfp_sub_double.vhd
nfp_add_double.vhd
nfp_mul_double.vhd
LPF.vhd
nfp_sqrt_double.vhd
AM_demodulator.vhd
Referenced Models
Clock Report for am_demod_hdl_coder
Rate Information
Model Base Rate 1e-06
DUT Base Rate 1e-06
Clock Enable Table
Clock Enable Name Sample Time
ce_out 1e-06
Output Signal Table
Output Signal Clock Enable Name Sample Time Latency
Audio_out ce_out 1e-06 0
LPF
View All
am_demod_hdl_coder
AM demodulator
LPF
Input
Main
Signal Attributes
OK
Help


Documents
am_git
AM_receiver_SDR
hdl_coder
hdl_prj
hdlsrc
am_demod_hdl_coder
Recent
Starred
Home
Desktop
Documents
Downloads
Music
Pictures
2022.2
am_demod_hdl_coder
Home
html
AM_demodulator.vhd
AM_demodulator_compi...
AM_demodulator_map.txt
AM_demodulator_repor...
AM_demodulator_tb.vhd
AM_demodulator_tb_co...
AM_demodulator_tb_pkg.vhd
AM_demodulator_tb_si...
annotate_port.m
clearhighlighting.m
criticalPathEstimated.m
criticalpathestimationsummary...
gm_am_demod_hdl_code...
gm_am_demod_hdl_code...
hcv
hdlcodegenstatus.mat
hdlwfbuildstatus.mat
LPF.vhd
nfp_add_double.vhd
nfp_gain_pow2_double.vhd
nfp_mul_double.vhd
nfp_sqrt_double.vhd
nfp_sub_double.vhd
tpfc036066_c594_46fb_a8d...

EMBEDDED CODER

Embedded Coder® is a product developed by MathWorks that enables the generation of readable, compact, and efficient C and C++ code specifically designed for embedded processors used in mass production. It extends the capabilities of MATLAB® Coder™ and Simulink® Coder by providing advanced optimizations that allow for precise control over the generated functions, files, and data.

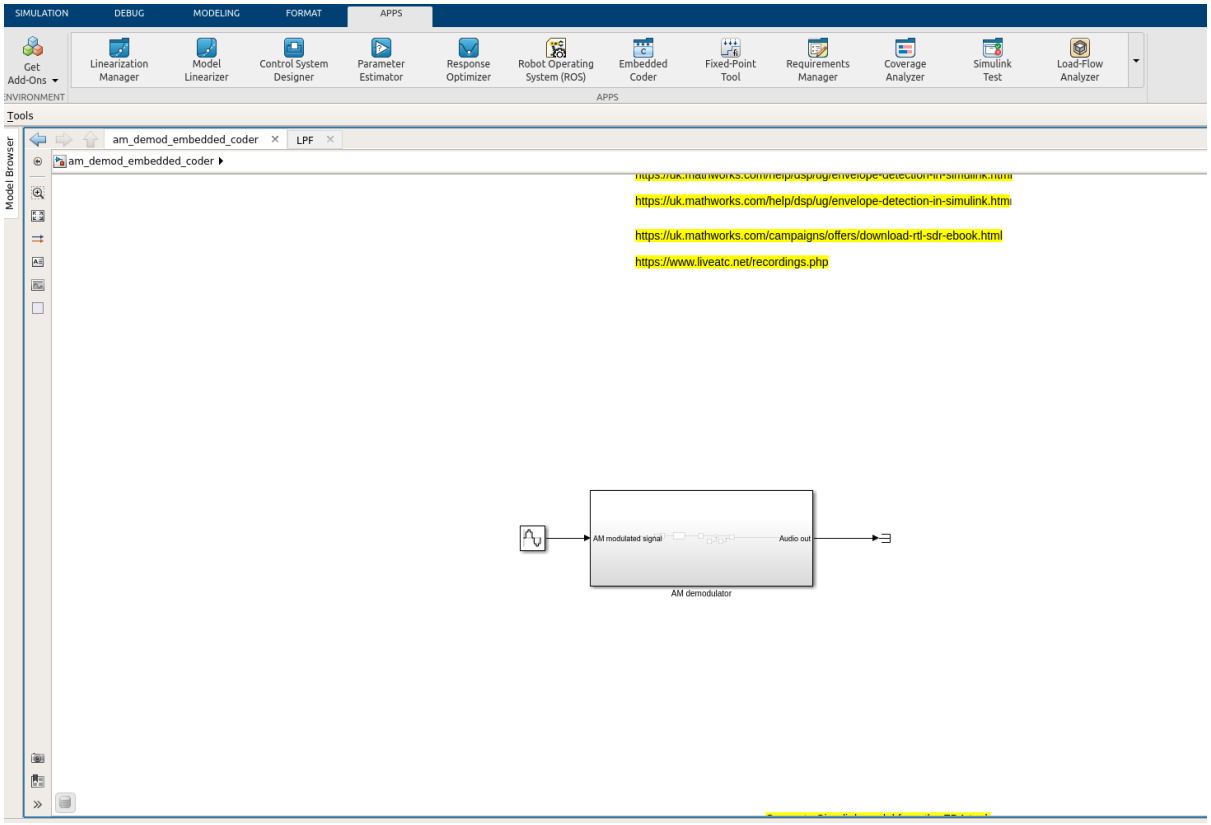
https://www.mathworks.com/products/embedded-coder.html

Its key features are:

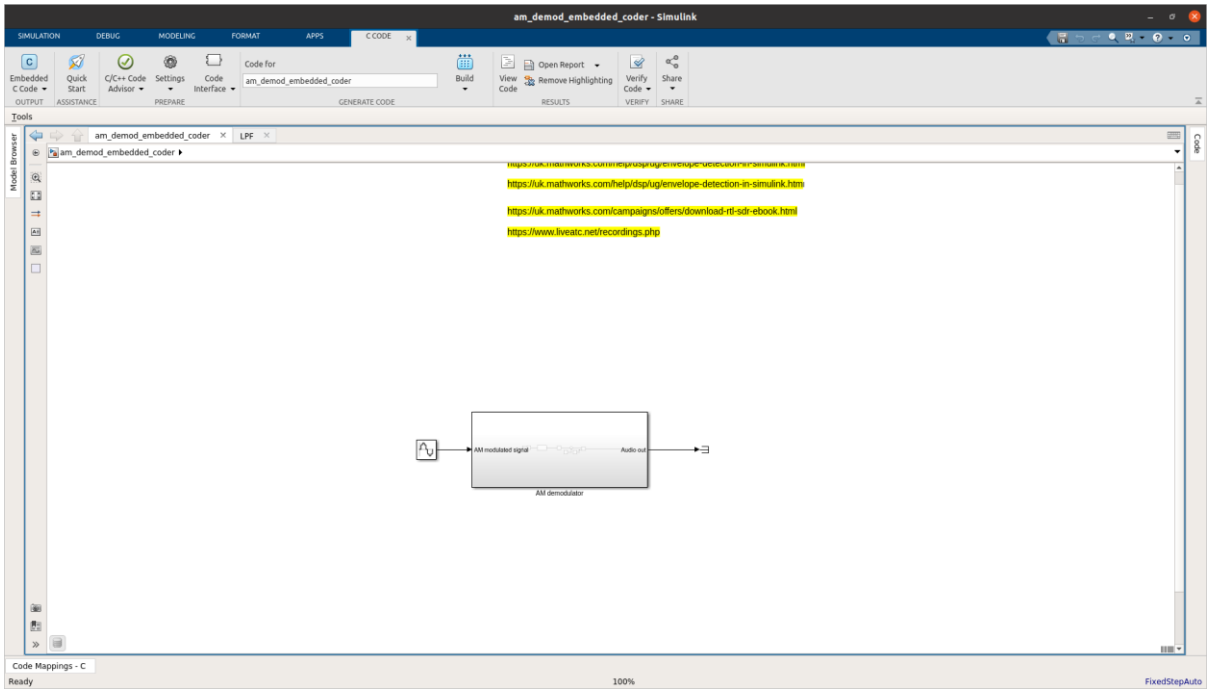
- Code Generation: Embedded Coder generates optimized code from models created in MATLAB and Simulink. This code can be tailored to meet specific requirements for embedded systems, ensuring that it is both efficient and portable.
- Optimization Techniques: The product incorporates various optimization techniques to enhance code efficiency. These include processor-specific optimizations such as SIMD (Single Instruction, Multiple Data), which can significantly improve performance on compatible hardware.
- Customization Options: Users can customize how models appear in the generated code functions and data. This feature facilitates software integration and helps satisfy coding preferences and standards, making it easier to work with legacy code or specific project requirements.
- Deployment Capabilities: Embedded Coder allows users to generate complete executables with input/output capabilities for popular hardware platforms.Testing and Verification: The tool supports software-in-the-loop (SIL) and processor-in-the-loop (PIL) testing methodologies to verify the correctness of generated code against model behavior. It also provides metrics for code coverage and execution profiles to ensure reliability.
- Compliance Support: Embedded Coder offers built-in support for various software standards such as AUTOSAR, MISRA C™, DO178, IEC 61508, and ISO 26262. This compliance is crucial for industries where safety-critical applications are developed.
- Documentation and Reporting: The product generates traceability reports that link generated code back to models and requirements, along with documentation that aids in understanding the structure and functionality of the produced code.

The C code has been generated in a similar way used for the HDL code.

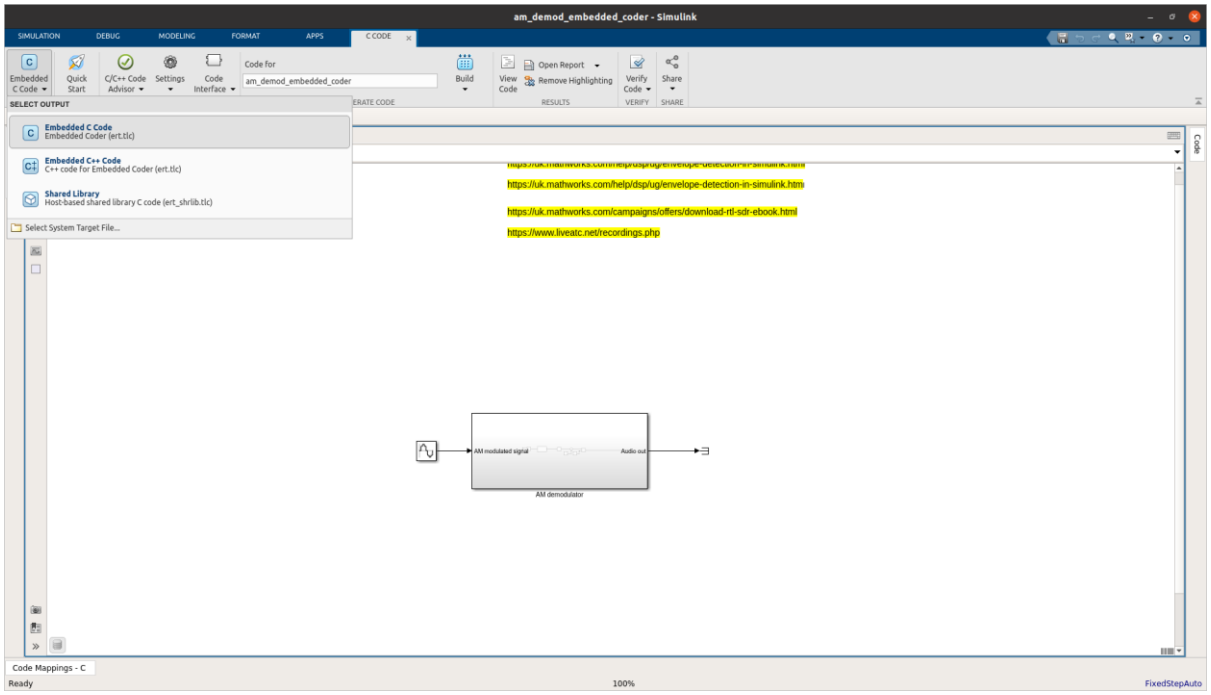
The starting model is:



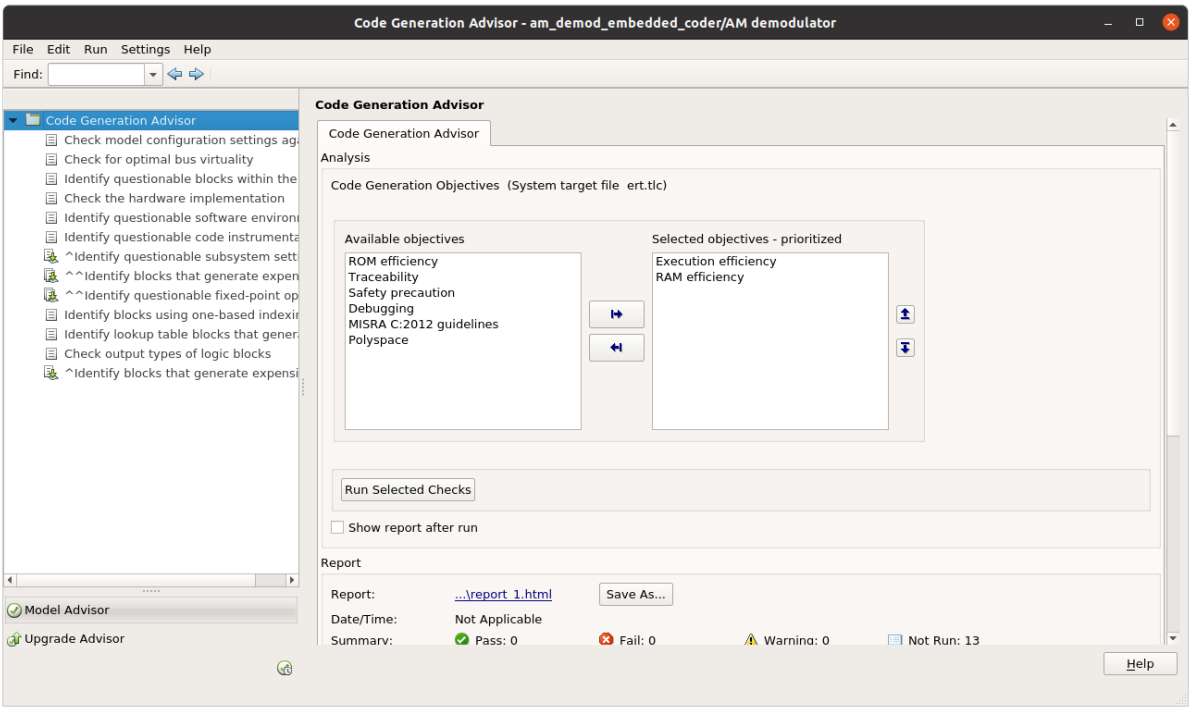
Invoking the embedded coder app:



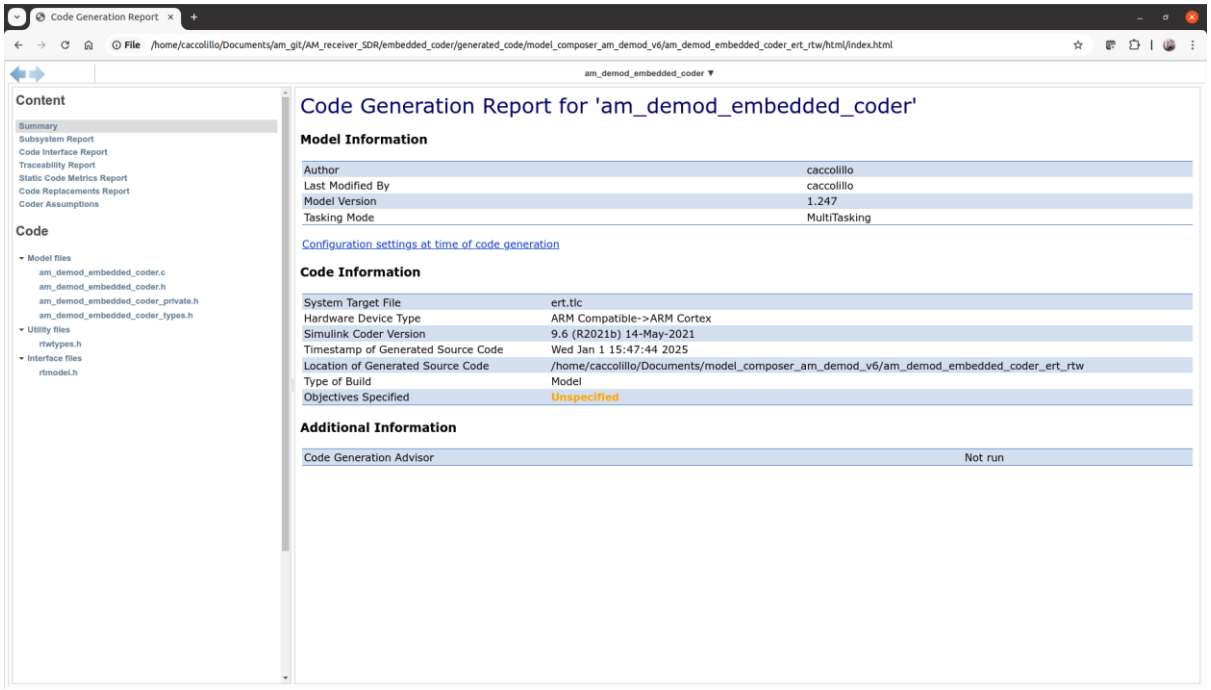
There are several options for generating the code:

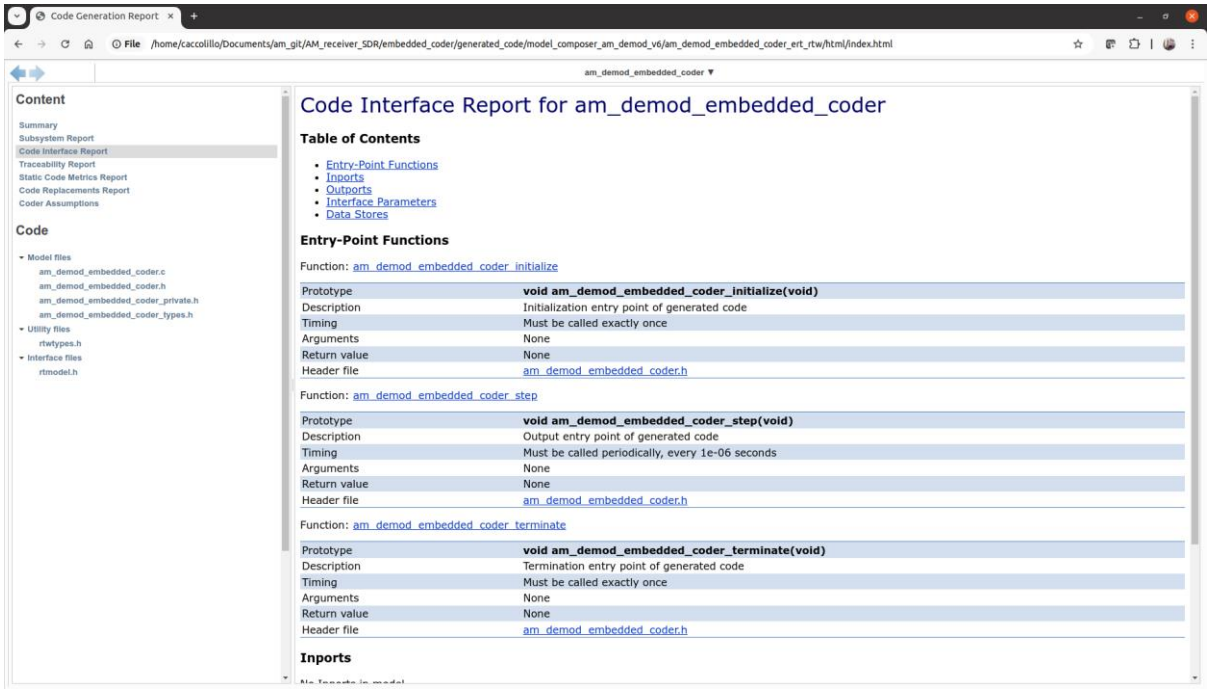


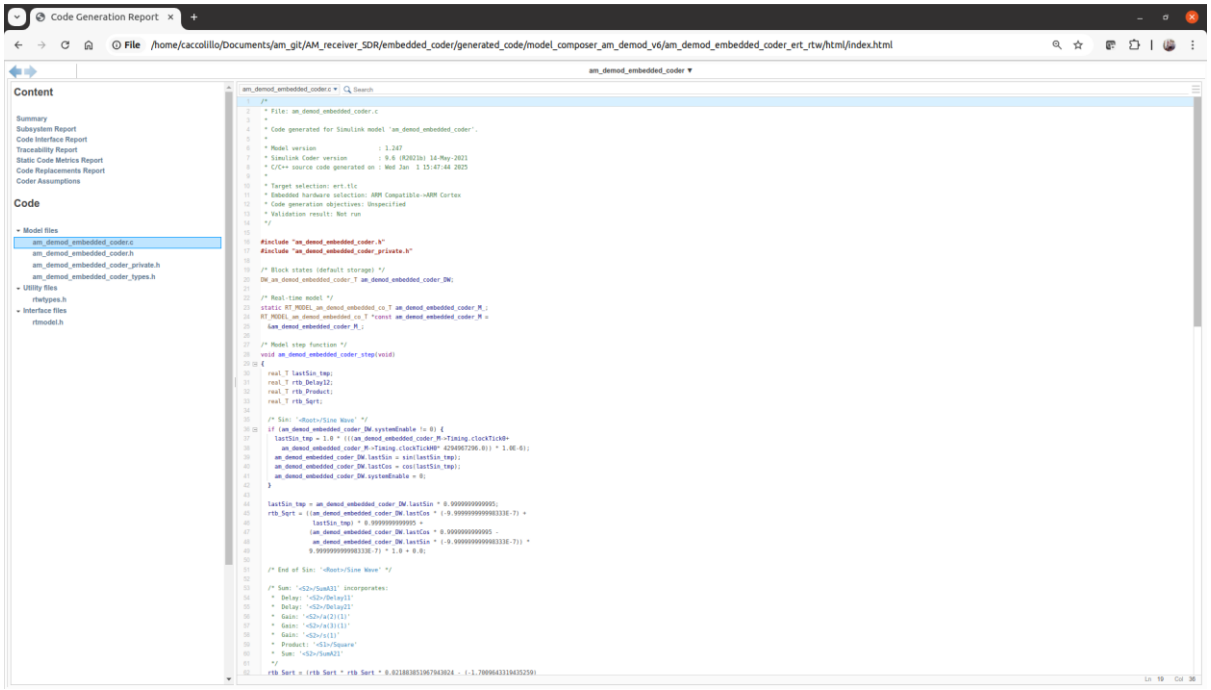
Code advisor is available:

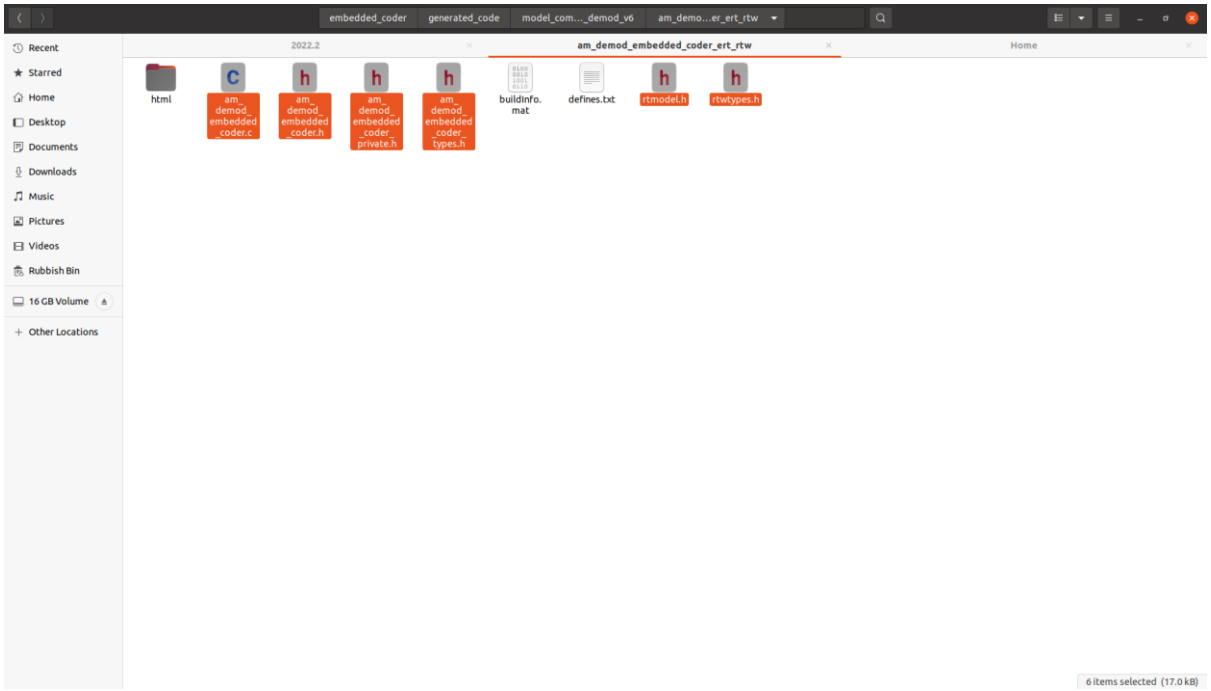


When the C code is generated, an in depth report is made available:


Code Generation Report
File /home/caccolillo/Documents/am_git/AM_receiver_SDR/embedded_coder/generated_code/model_composer_am_demod_v6/am_demod_embedded_coder_ert_rtw/html/index.html
am_demod_embedded_coder
Content
Summary
Subsystem Report
Code Interface Report
Traceability Report
Static Code Metrics Report
Code Replacements Report
Coder Assumptions
Code
Model files
am_demod_embedded_coder.c
am_demod_embedded_coder.h
am_demod_embedded_coder_private.h
am_demod_embedded_coder_types.h
Utility files
rtwtypes.h
Interface files
rtmodel.h
Code Generation Report for 'am_demod_embedded_coder'
Model Information
Author caccolillo
Last Modified By caccolillo
Model Version 1.247
Tasking Mode MultiTasking
Configuration settings at time of code generation
Code Information
System Target File ert.tlc
Hardware Device Type ARM Compatible->ARM Cortex
Simulink Coder Version 9.6 (R2021b) 14-May-2021
Timestamp of Generated Source Code Wed Jan 1 15:47:44 2025
Location of Generated Source Code /home/caccolillo/Documents/model_composer_am_demod_v6/am_demod_embedded_coder_ert_rtw
Type of Build Model
Objectives Specified Unspecified
Additional Information
Code Generation Advisor Not run


Code Generation Report
File /home/caccolillo/Documents/am_git/AM_receiver_SDR/embedded_coder/generated_code/model_composer_am_demod_v6/am_demod_embedded_coder_ert_rtw/html/index.html
am_demod_embedded_coder
Content
Summary
Subsystem Report
Code Interface Report
Traceability Report
Static Code Metrics Report
Code Replacements Report
Coder Assumptions
Code
Model files
am_demod_embedded_coder.c
am_demod_embedded_coder.h
am_demod_embedded_coder_private.h
am_demod_embedded_coder_types.h
Utility files
rtwtypes.h
Interface files
rtmodel.h
Code Interface Report for am_demod_embedded_coder
Table of Contents
Entry-Point Functions
Inports
Outports
Interface Parameters
Data Stores
Entry-Point Functions
Function: am_demod_embedded_coder_initialize
Prototype void am_demod_embedded_coder_initialize(void)
Description Initialization entry point of generated code
Timing Must be called exactly once
Arguments None
Return value None
Header file am_demod_embedded_coder.h
Function: am_demod_embedded_coder_step
Prototype void am_demod_embedded_coder_step(void)
Description Output entry point of generated code
Timing Must be called periodically, every 1e-06 seconds
Arguments None
Return value None
Header file am_demod_embedded_coder.h
Function: am_demod_embedded_coder_terminate
Prototype void am_demod_embedded_coder_terminate(void)
Description Termination entry point of generated code
Timing Must be called exactly once
Arguments None
Return value None
Header file am_demod_embedded_coder.h
Inports





MODEL COMPOSER

Model Composer is a tool developed by MathWorks that facilitates model-based design for embedded systems. It allows engineers and developers to create, simulate, and deploy algorithms within the Simulink environment. The primary purpose of Model Composer is to streamline the design process for complex systems, particularly those that require high-performance digital signal processing (DSP).

Model Composer provides a graphical interface where users can create block diagrams representing their system architectures. This visual approach simplifies the modelling process and makes it easier to understand complex interactions between components.

Users can simulate their designs early in the development cycle, allowing for testing and validation before moving to hardware implementation. This feature helps identify potential issues and optimize performance.

The tool integrates seamlessly with MATLAB , Simulink and Xilinx Vivado, enabling users to leverage existing models and algorithms while also providing access to an extensive library of bit- and cycle-accurate models.

One of the standout features of Model Composer is its ability to automatically generate production-quality code. This code can be deployed directly onto various hardware platforms, including AMD Versal™, RFSoC, and Zynq™-7000.

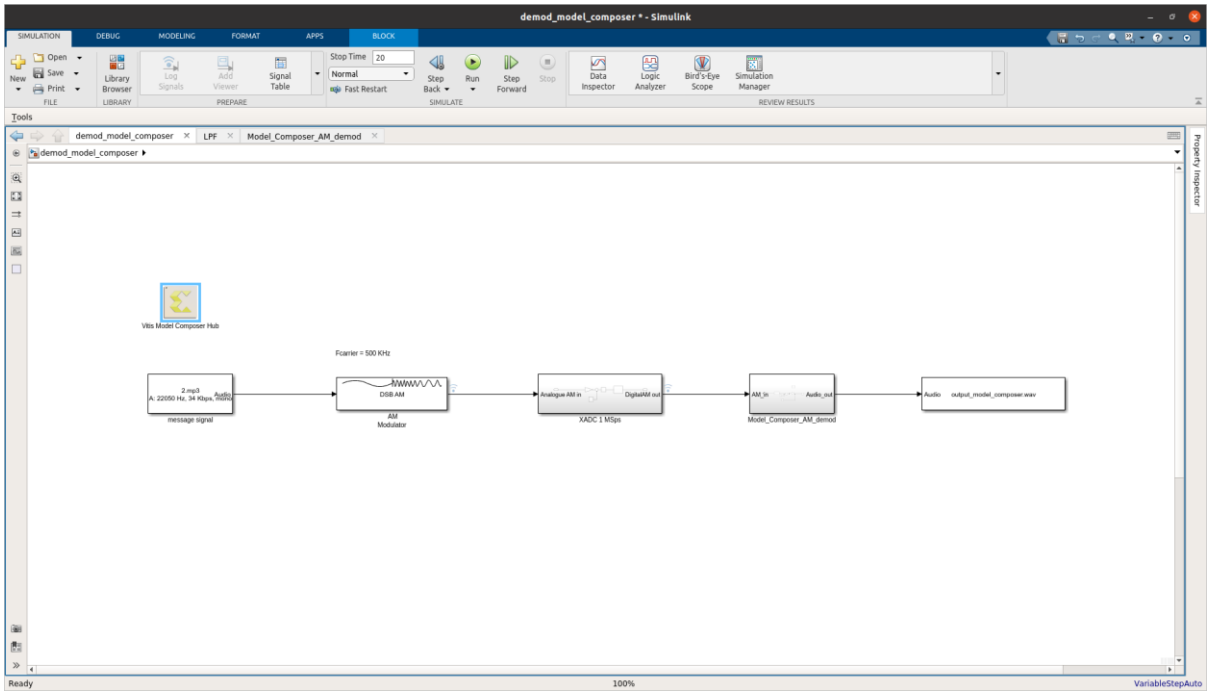
Users can import custom High-Level Synthesis (HLS), AI Engine, and Register Transfer Level (RTL) code as Intellectual Property (IP) blocks into their designs, enhancing flexibility and reusability.

Model Composer supports rapid prototyping through FPGA-in-the-loop testing, allowing developers to validate their designs against physical models.

Simulations in Model Composer are bit-true and cycle-true. To say a simulation is bit-true means that at the boundaries (i.e., interfaces between System Generator HDL blocks and non-System Generator HDL blocks), a value produced in simulation is bit-for-bit identical to the corresponding value produced in hardware. To say a simulation is cycle-true means that at the boundaries, corresponding values are produced at corresponding times.

https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-model-composer.html

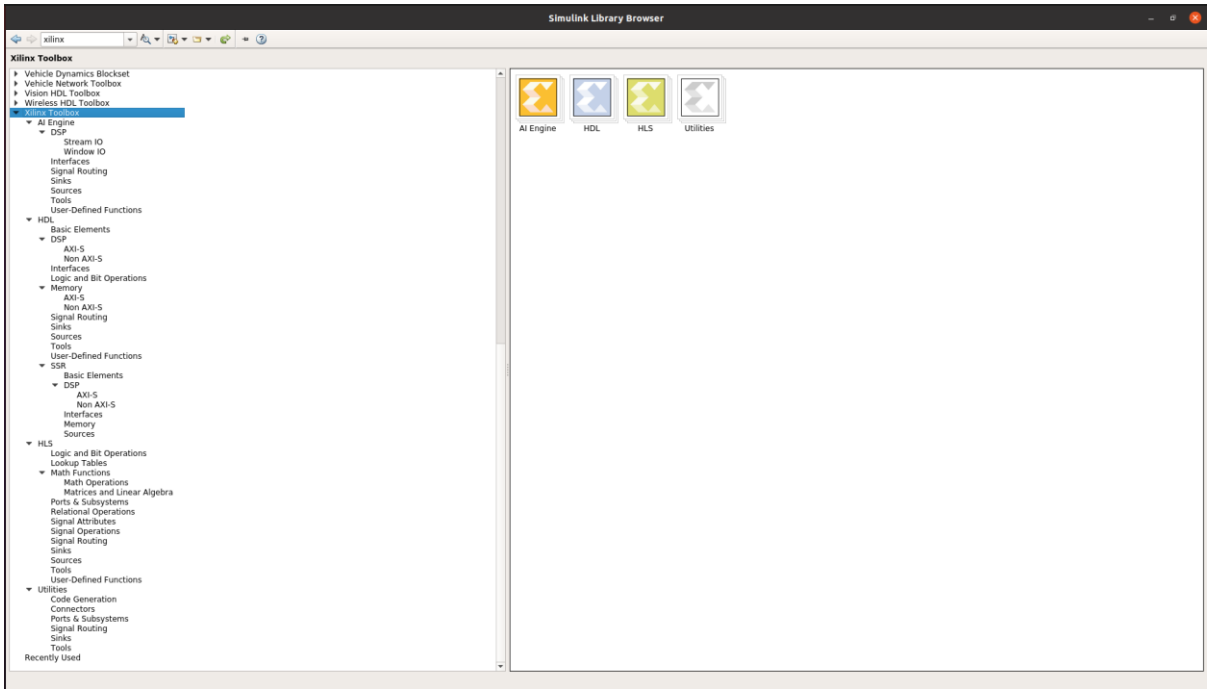
The model made by using model composer is:

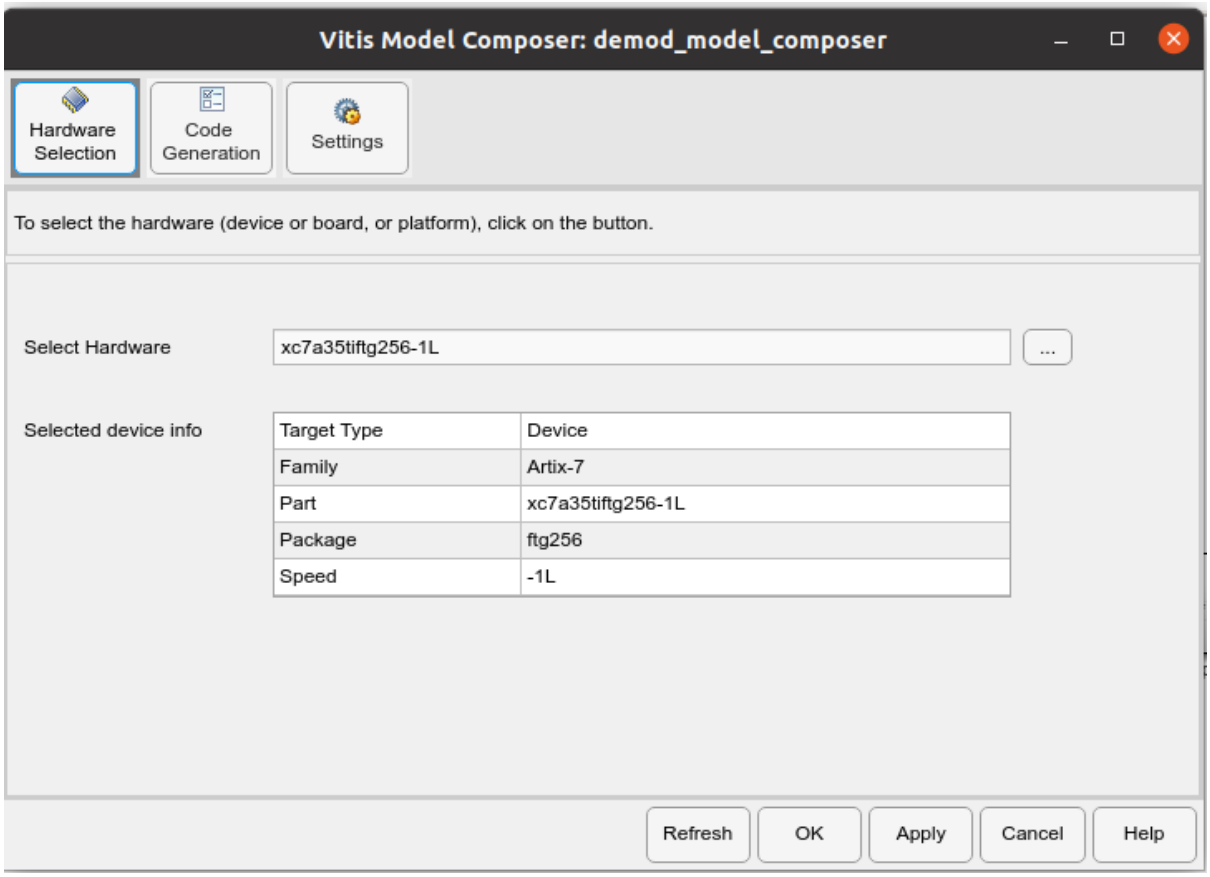


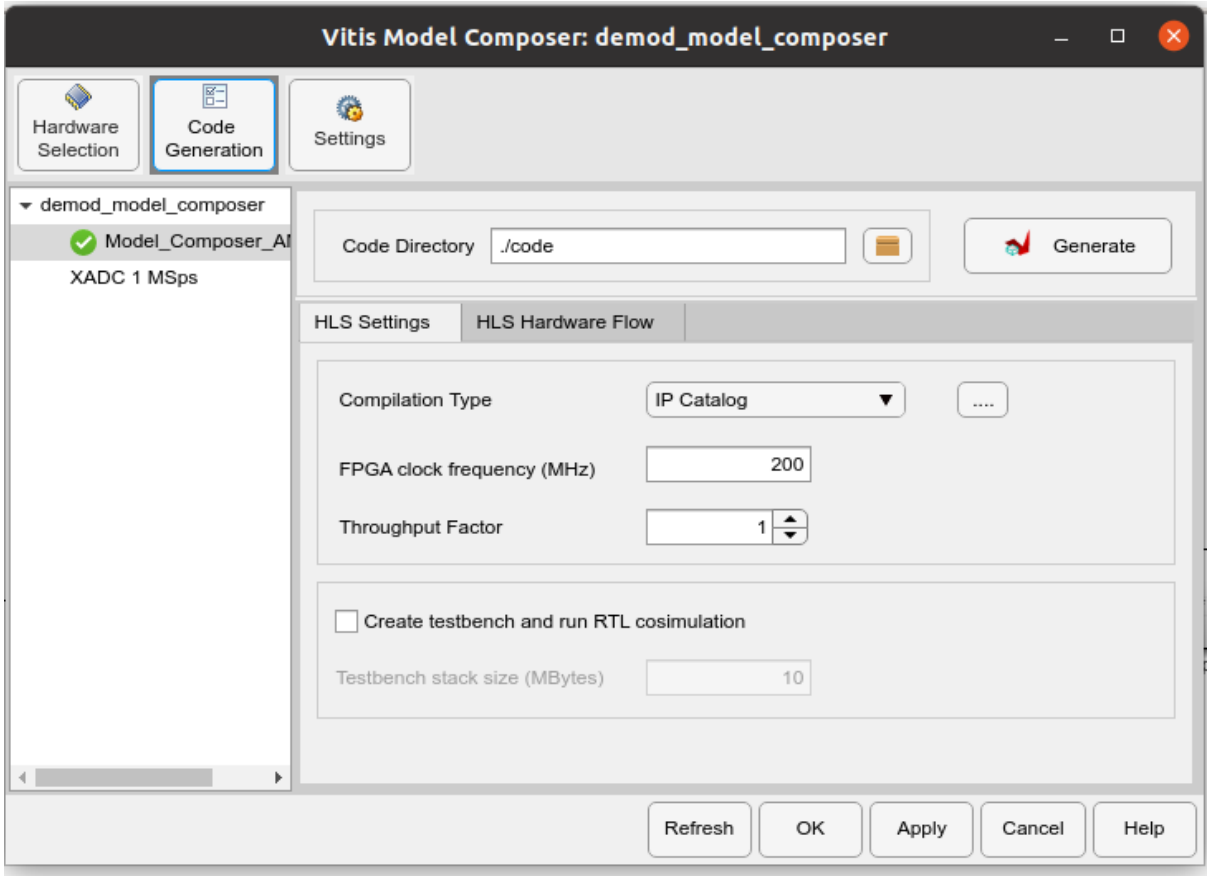
Being a tool suited for Xilinx devices only, it stands at a little bit lower abstraction layer than HDL coder.

In the library of available blocks, there's plenty of Xilinx specific low level primitives:

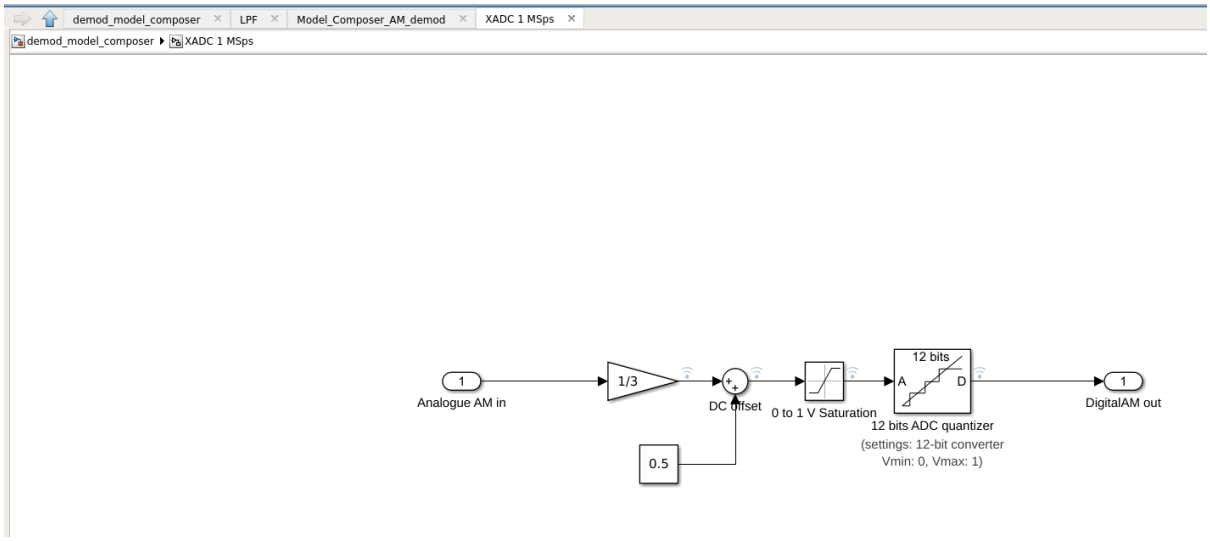
https://www.youtube.com/watch?v=nMfYF4daxaI



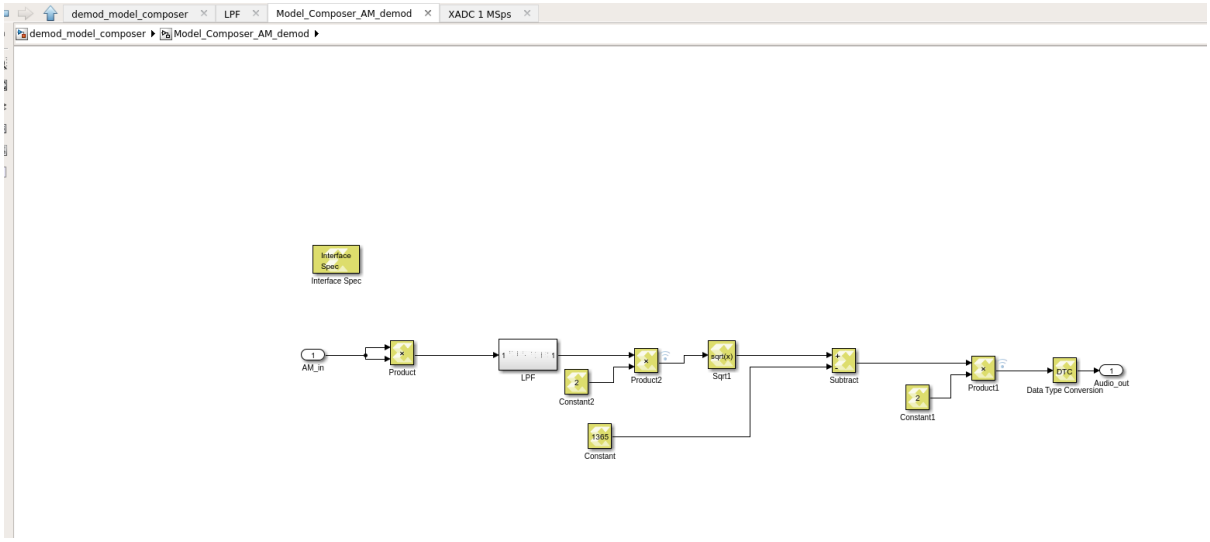


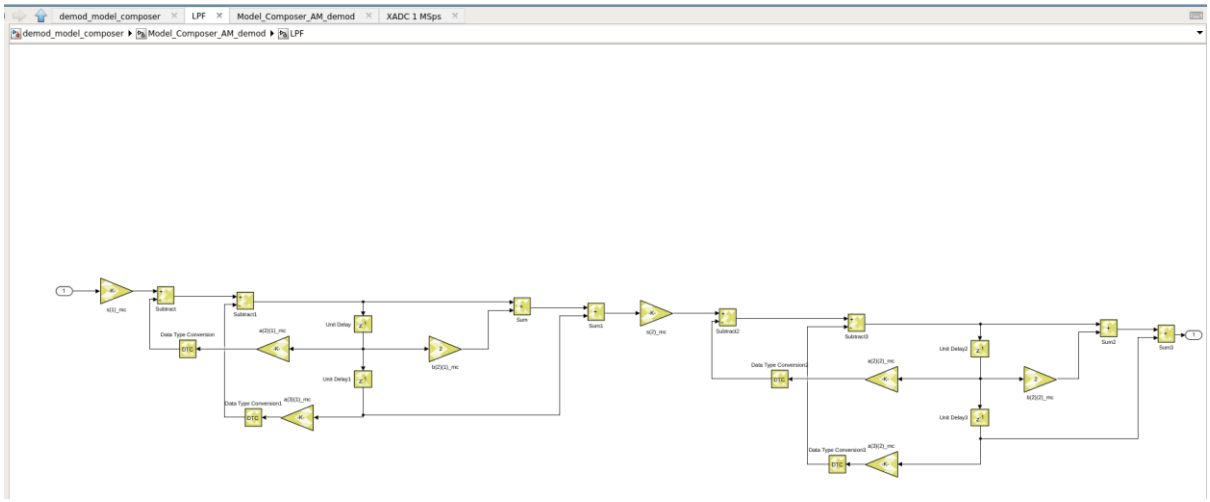


In this model, a rough modelling of the effects of the XADC have been introduced:

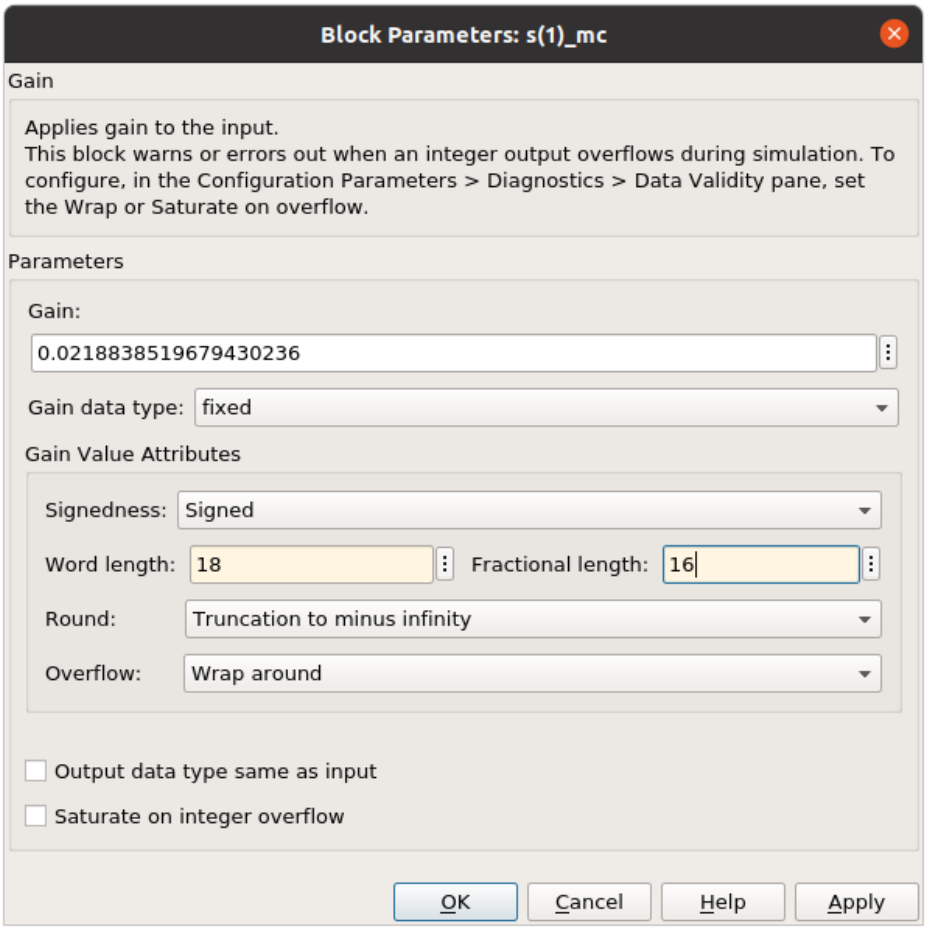


By having a look at the AM demodulator, we see:





Each block can be configured in terms of fixed point precision, so to reflect what has been found when fixed point designer has been used:



https://github.com/Xilinx/Vitis_Model_Composer/tree/2024.2/Tutorials

Once the model is generated, it can be exported as an IP-core to use in Vivado, or can be validated on hardware via HIL (hardware in the loop):

https://www.youtube.com/watch?v=8vbf9MZMOCY

https://docs.amd.com/v/u/2018.3-English/ug1262-model-composer-user-guide

Setting up HIL Test System

Simon's Team
HIL Systems Engineering

Powered actuators

Electronic loads

Power Supplies