

## 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:

$$s(t) = A_C [1 + \mu m(t)] \cos(2\pi f_c t)$$

where:

- AC is the amplitude of the carrier wave,
- $\mu$  is the modulation index,
- $m(t)$  is the message signal,
- $f_c$  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:

$$s^2(t) = [A_C (1 + \mu m(t)) \cos(2\pi f_c t)]^2$$

Expanding this expression yields:

$$s^2(t) = A_C^2 (1 + 2\mu m(t) + \mu^2 m^2(t)) \frac{1}{2} + A_C^2 \mu m(t) \cos(4\pi f_c t)$$

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

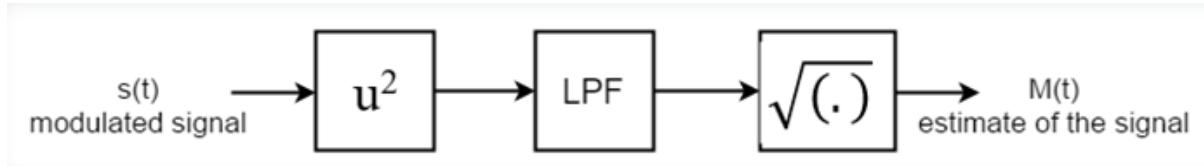
- A constant term,
- A term proportional to  $m(t)$ ,
- A high-frequency component at  $4f_c$

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:

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

K<sub>a</sub> 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](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>

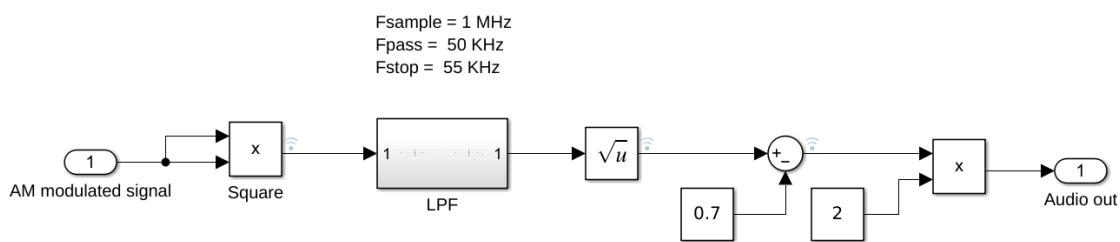
## SIMULINK GOLDEN MODEL

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

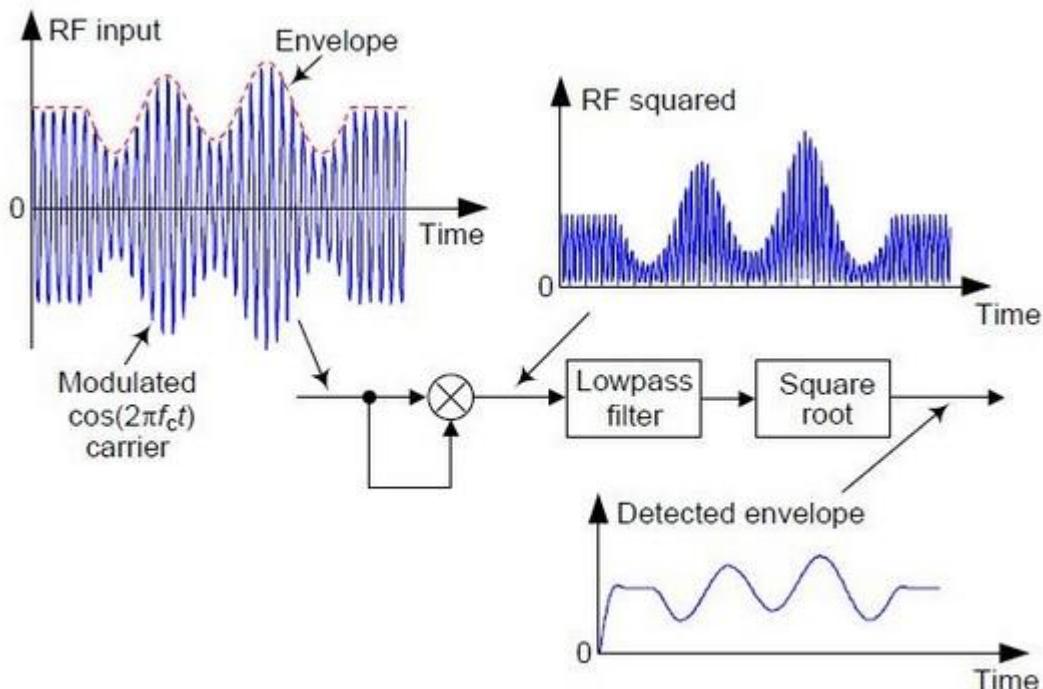
The golden model is typically characterized by its accuracy and completeness, ensuring that it represents the intended functionality of the system being modelled.

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.

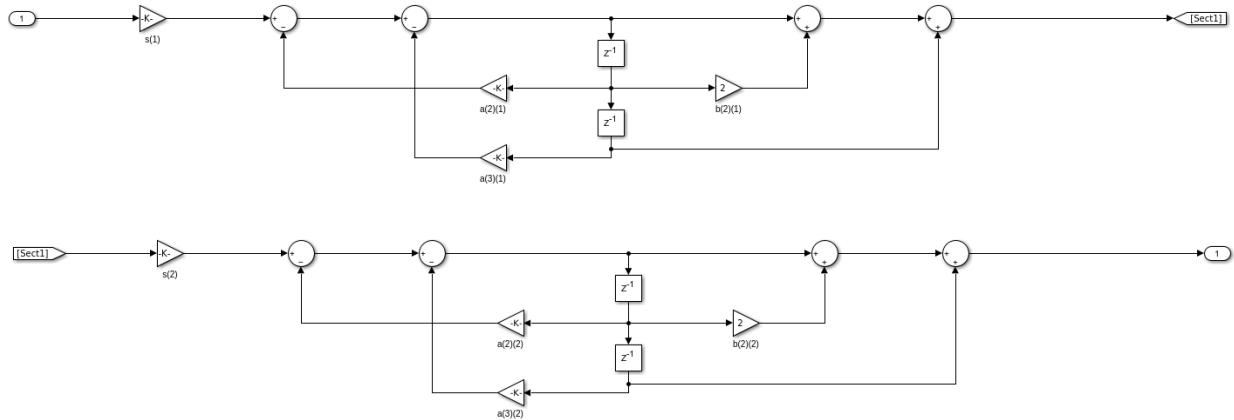
The golden model in Simulink:



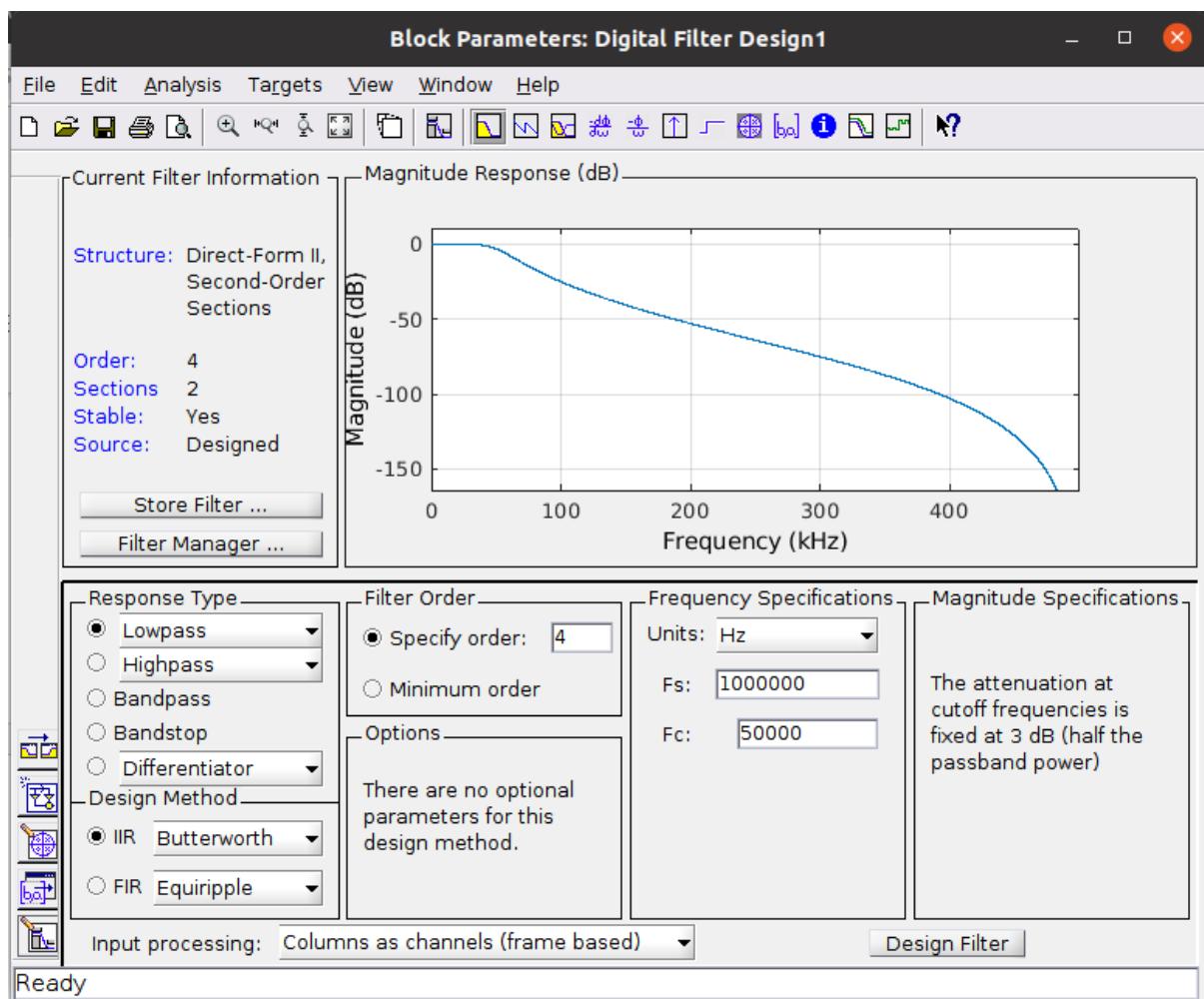
closely resembles the theoretical block diagram:

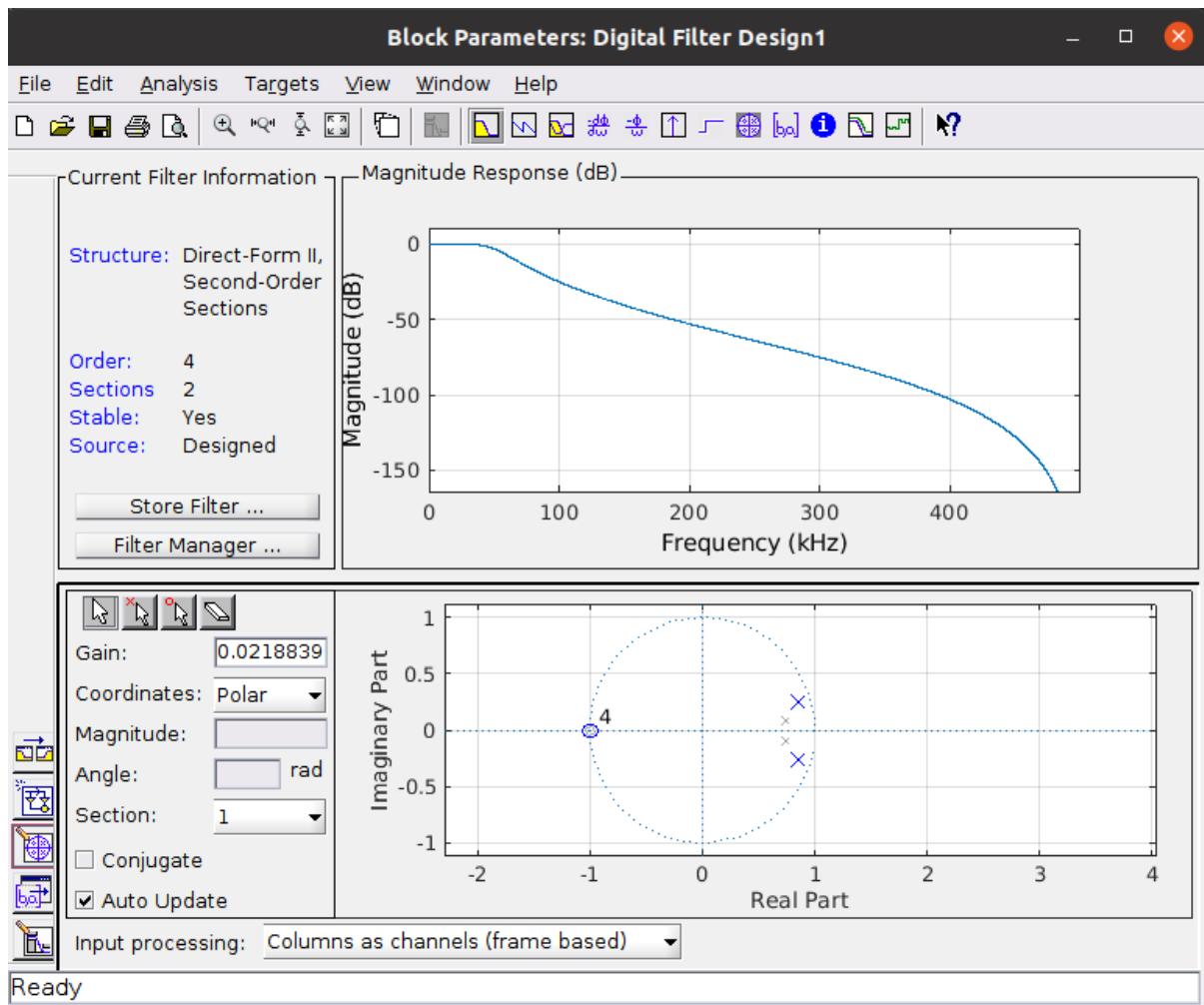


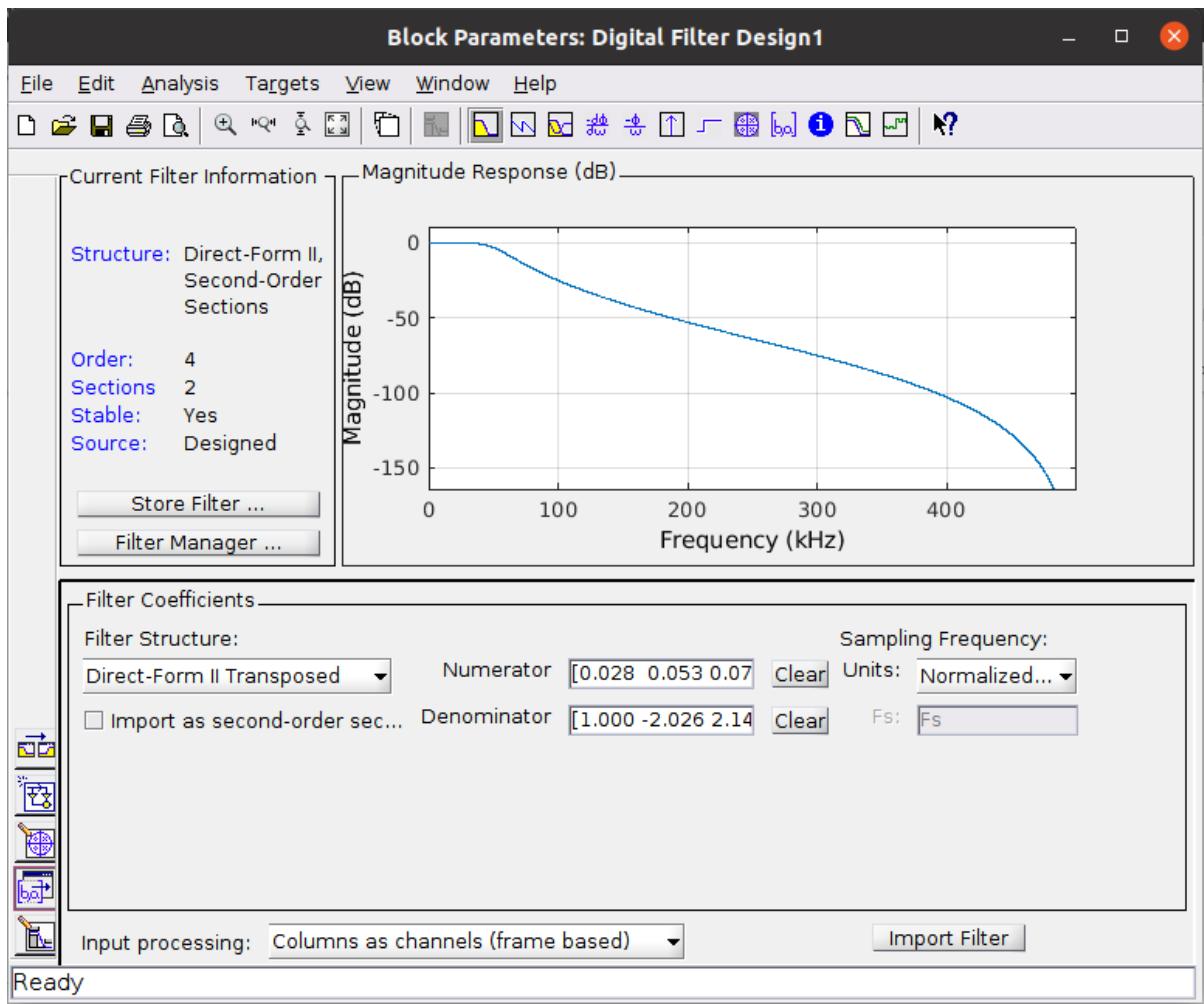
The LPF has been modelled as follows:



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







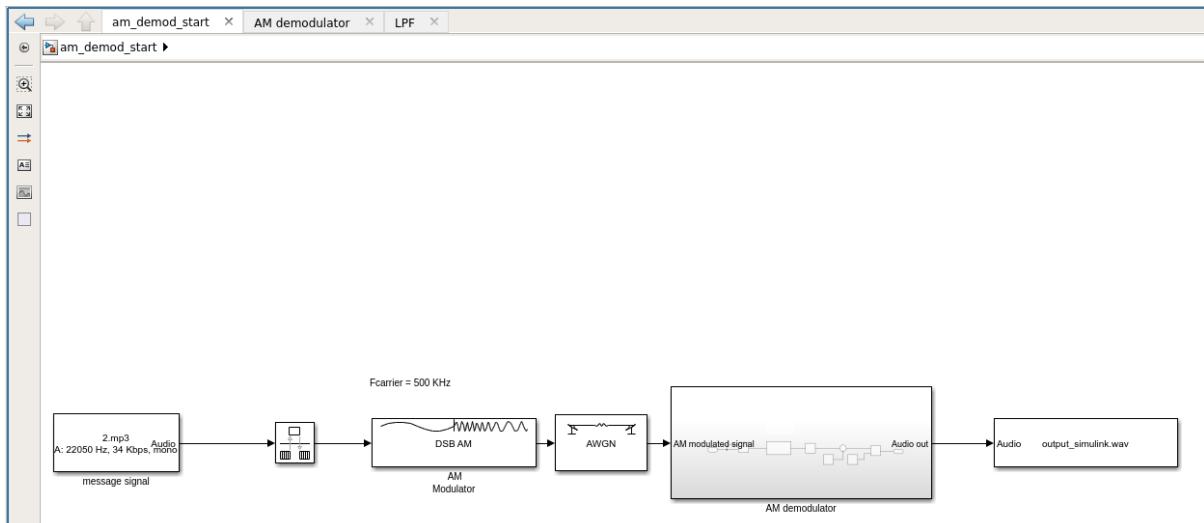
The DSP System Toolbox™ is a specialized software package designed for the design, simulation, and analysis of digital signal processing (DSP) systems. It is integrated with MATLAB® and Simulink®, providing a comprehensive environment for engineers and researchers to develop real-time signal processing 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:

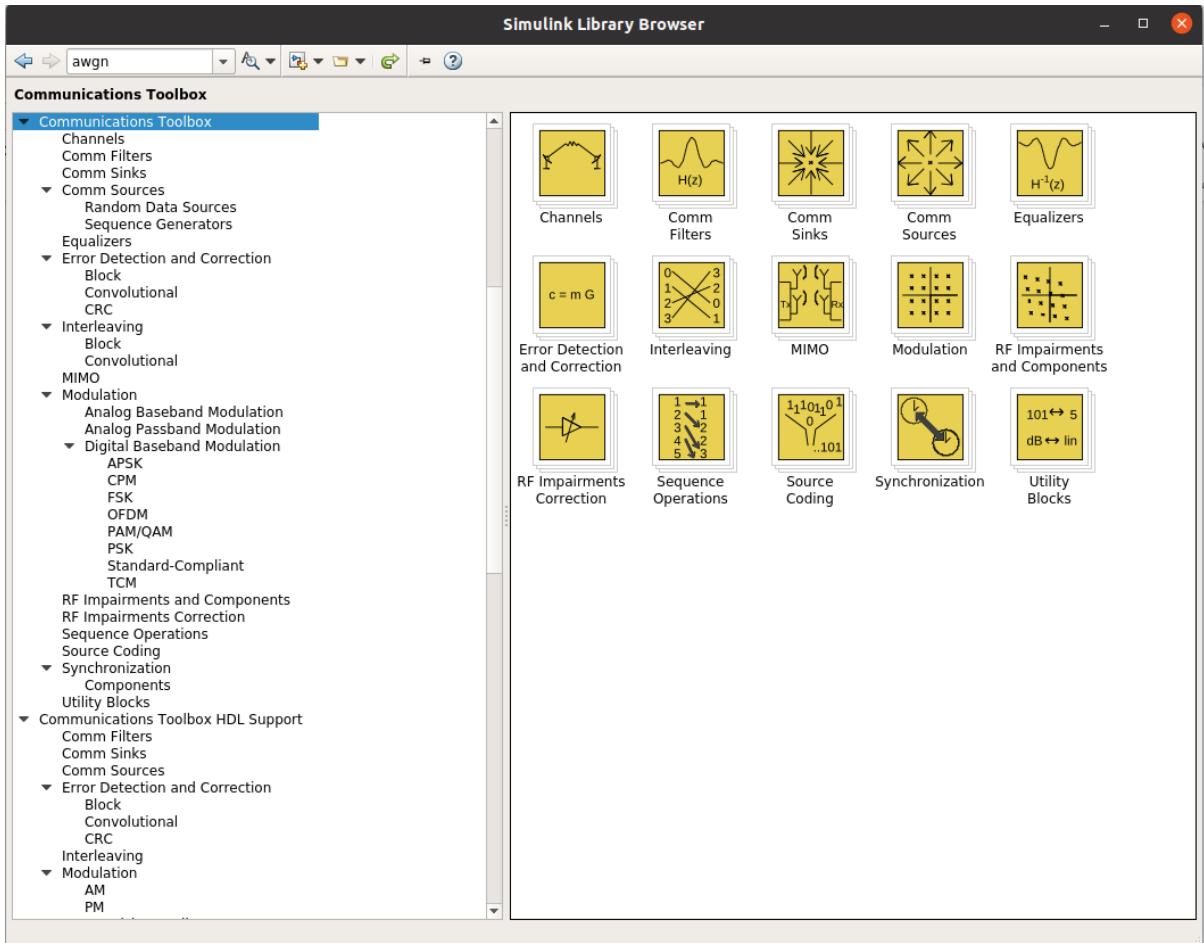
<https://www.liveatc.net/recording.php>

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.

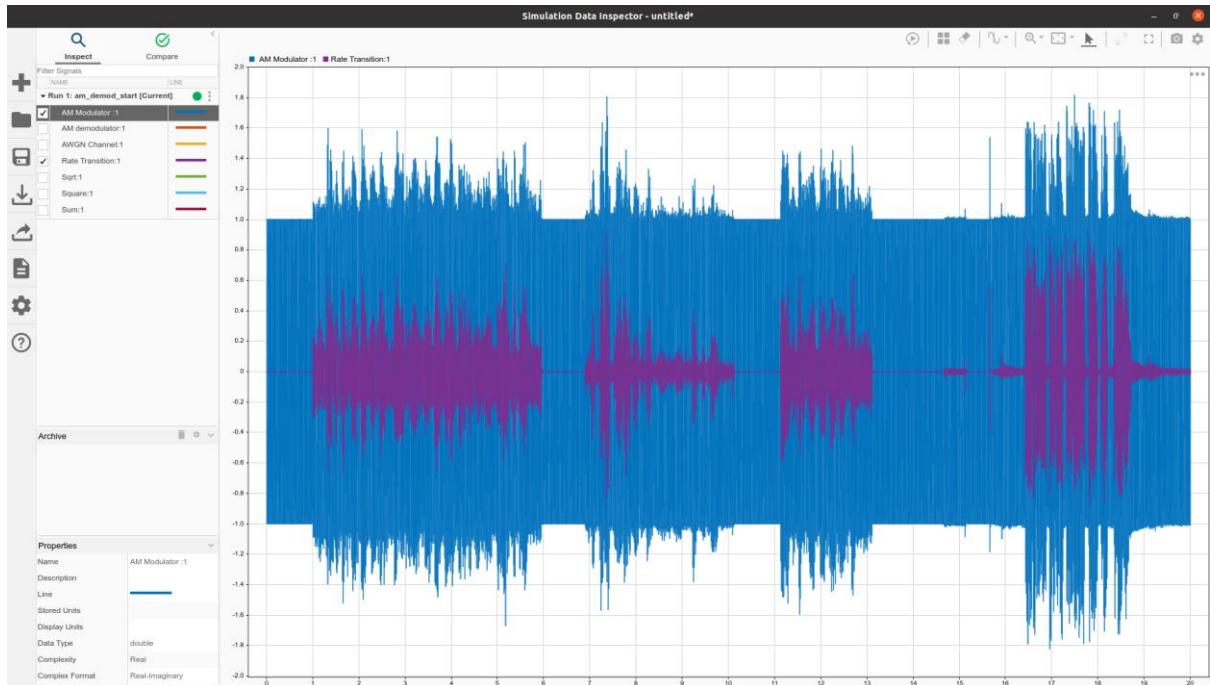


<https://uk.mathworks.com/products/communications.html>

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.

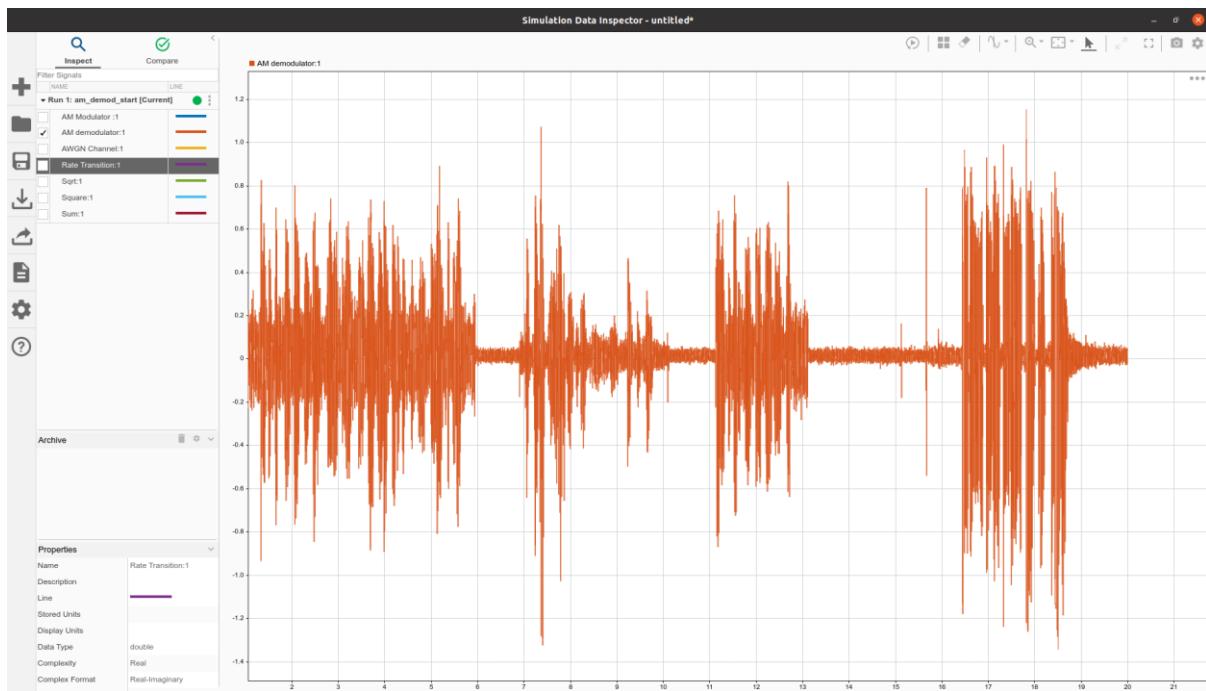
- Key Features are:
- Waveform Generation
- Receiver Performance Verification
- Artificial Intelligence Applications
- Propagation Channel Modeling
- Channel Degradation Compensation
- Software-Defined Radio (SDR) Testing
- Link-Level Simulation
- AI Techniques for Wireless Challenges
- Modeling RF Components
- 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:

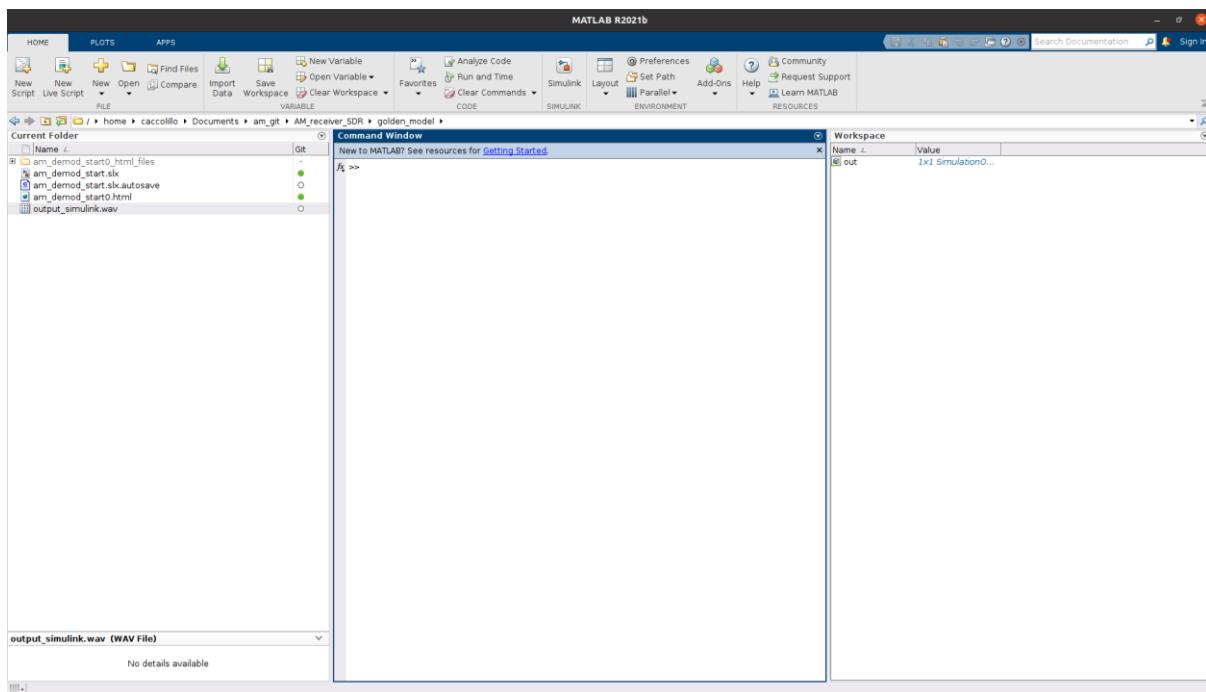


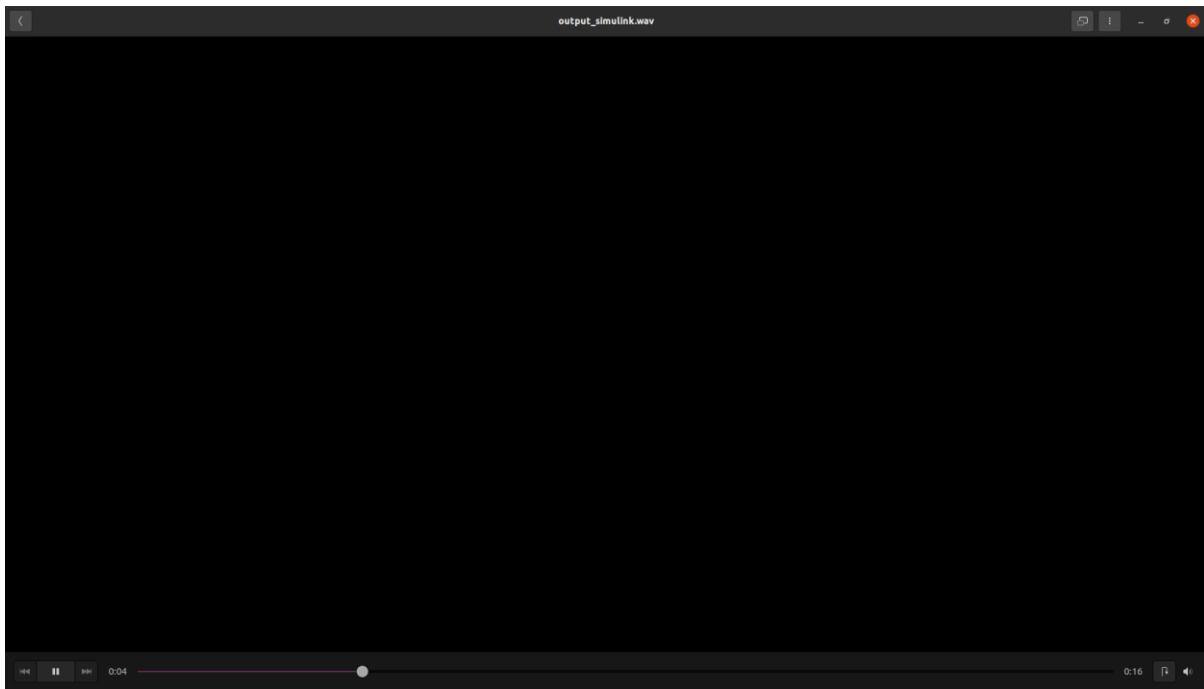
In purple there's the audio stream, and in blue there's the AM modulated input signal to the golden model.

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



Or can be listened too:



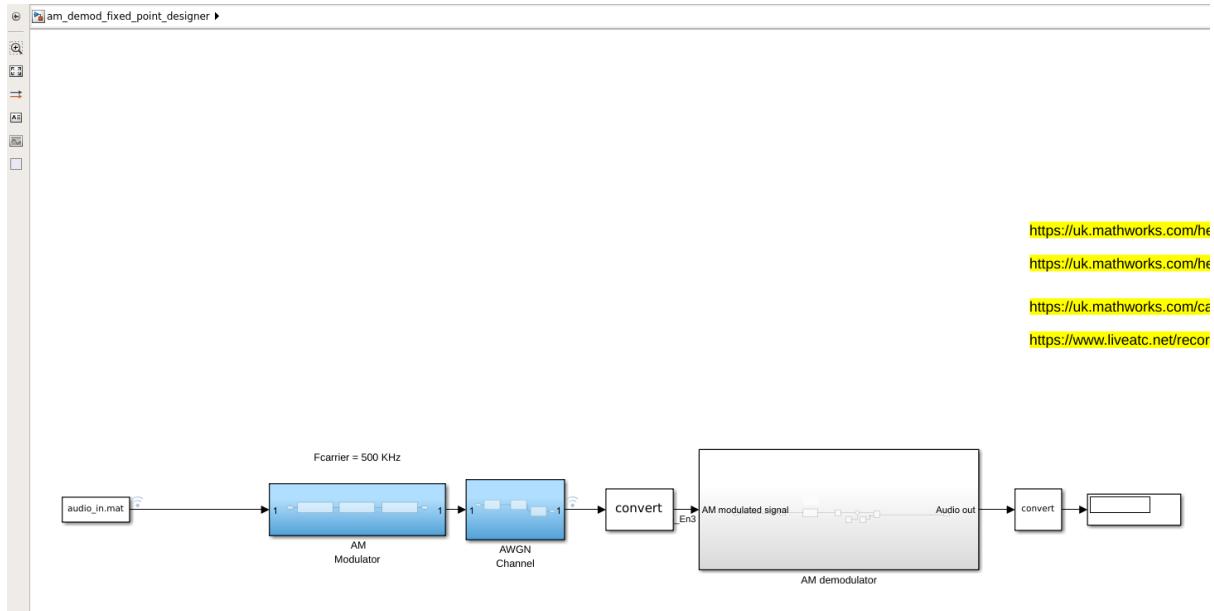


## FIXED POINT CONVERSION

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.

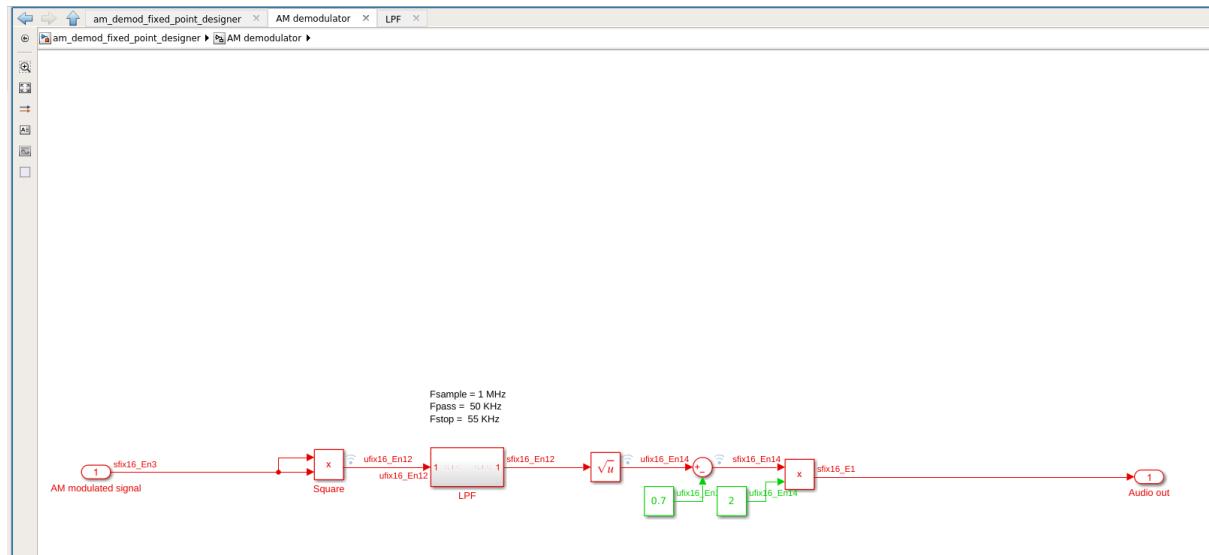
<https://uk.mathworks.com/products/fixed-point-designer.html>

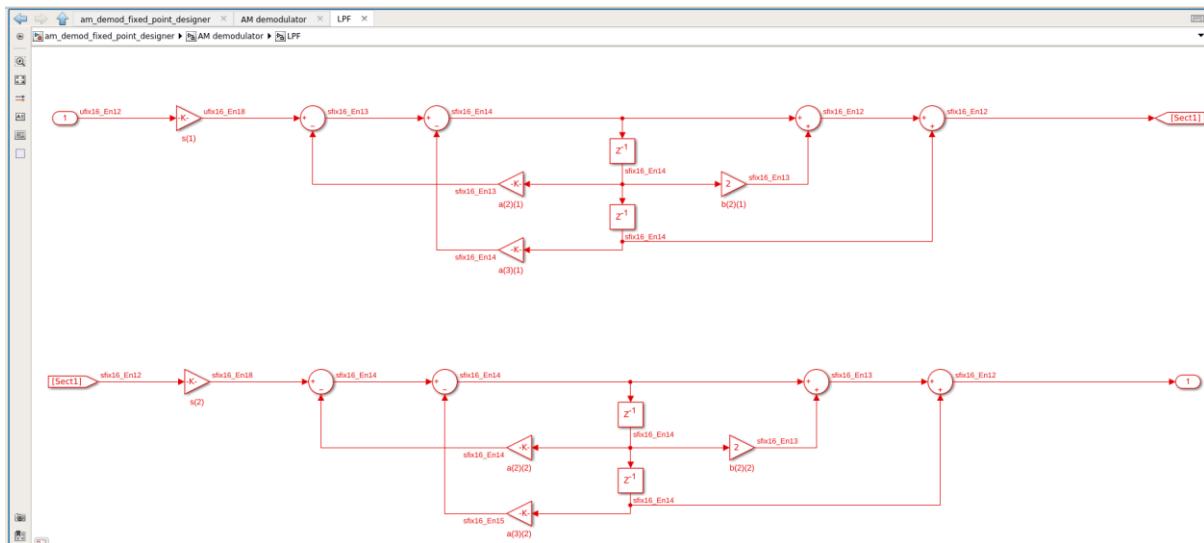
The floating point golden model, has been converted in fixed point thanks to this tool.



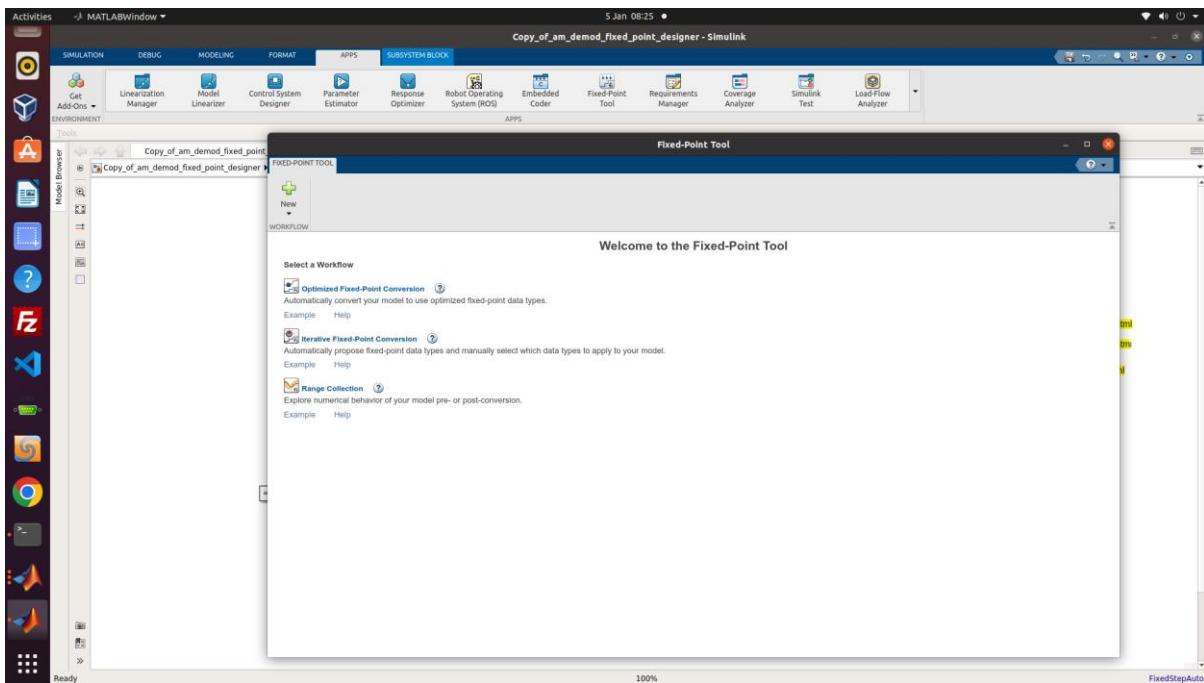
The block diagram is a minor change from the golden model, where the blocks involving reading and writing over audio files have been removed, because not compatible with fixed point designer.

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

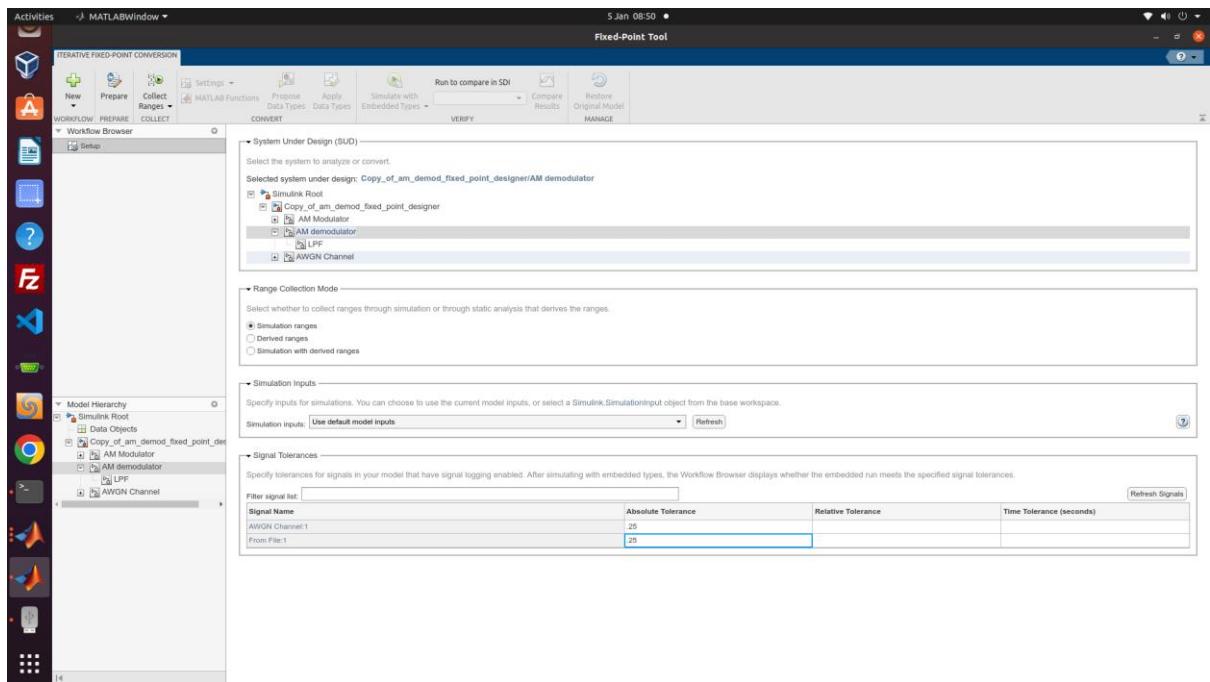




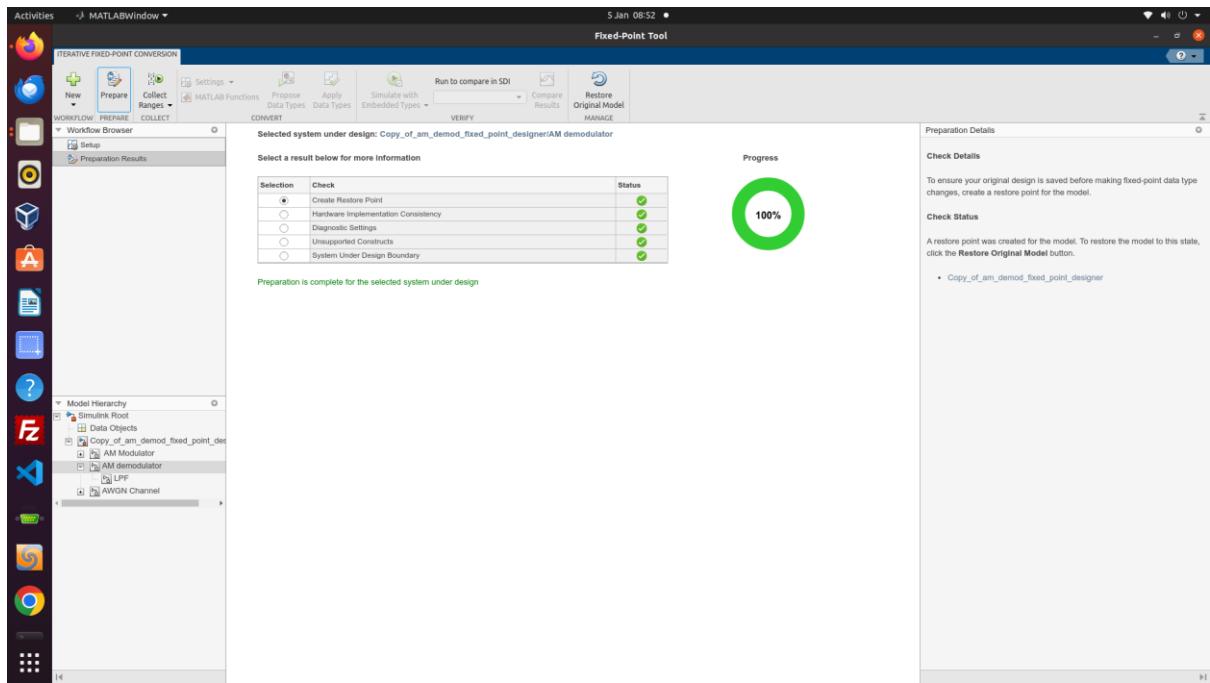
Starting from the floating point golden model block design, fixed point designer tool gets started:



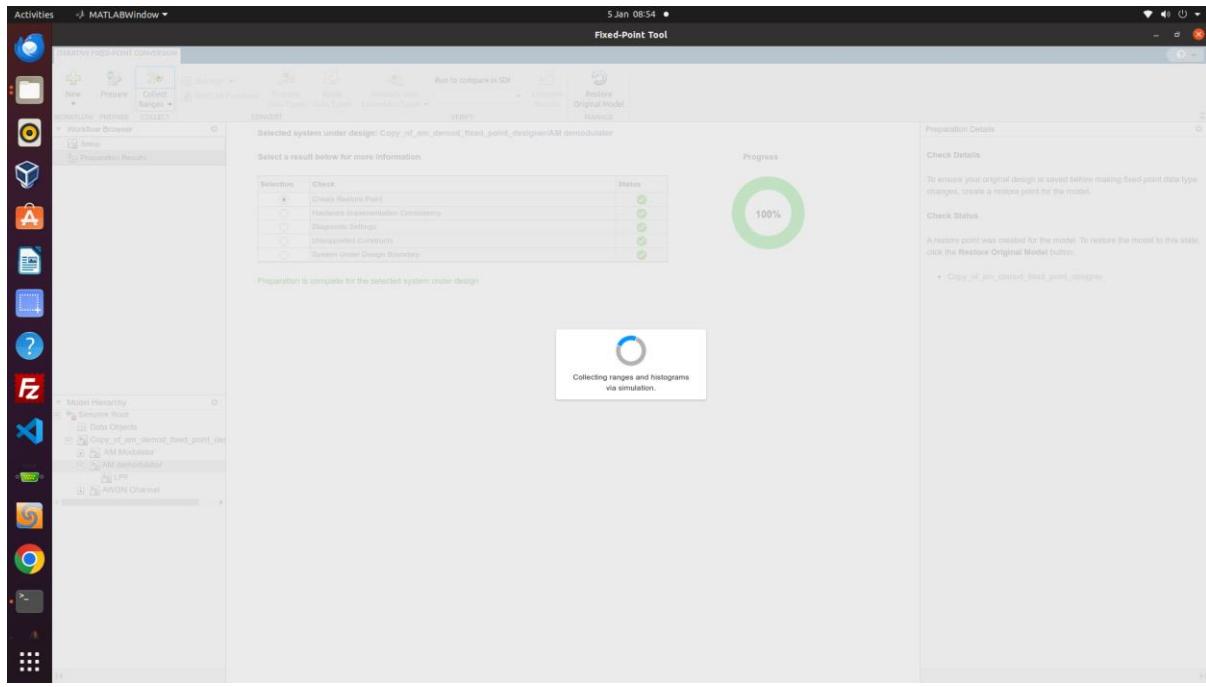
The iterative fixed point conversion option gets selected:



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



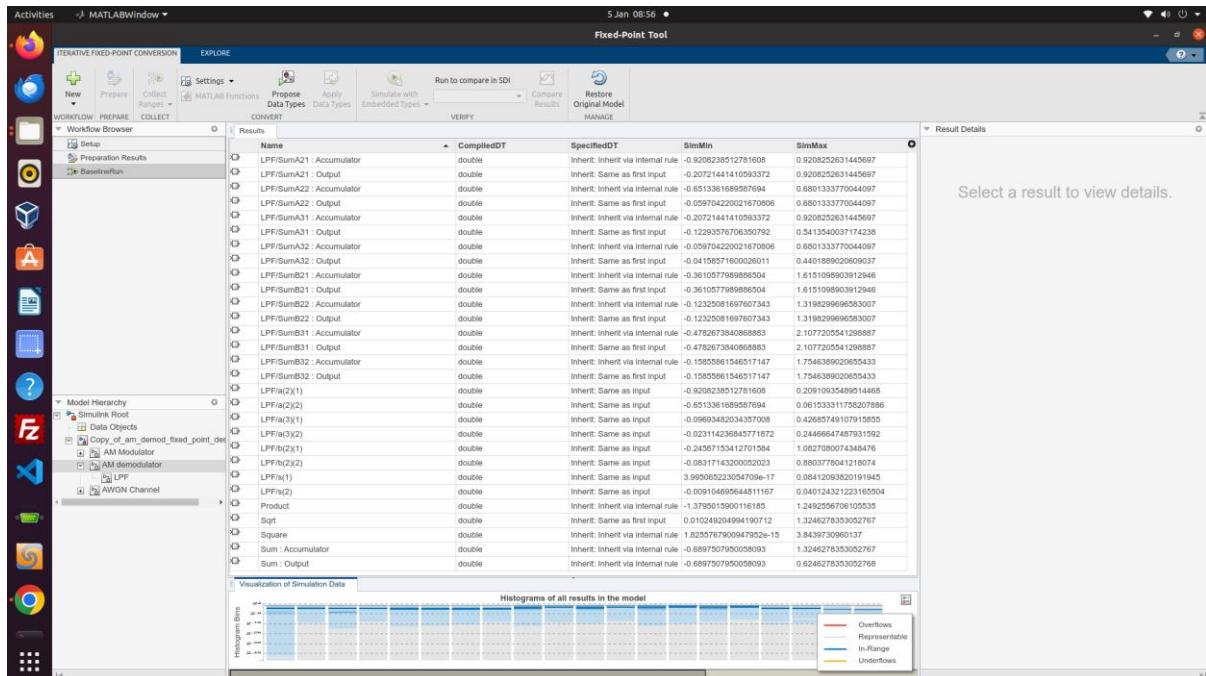
Then the “collect ranges” in double precision option gets selected:



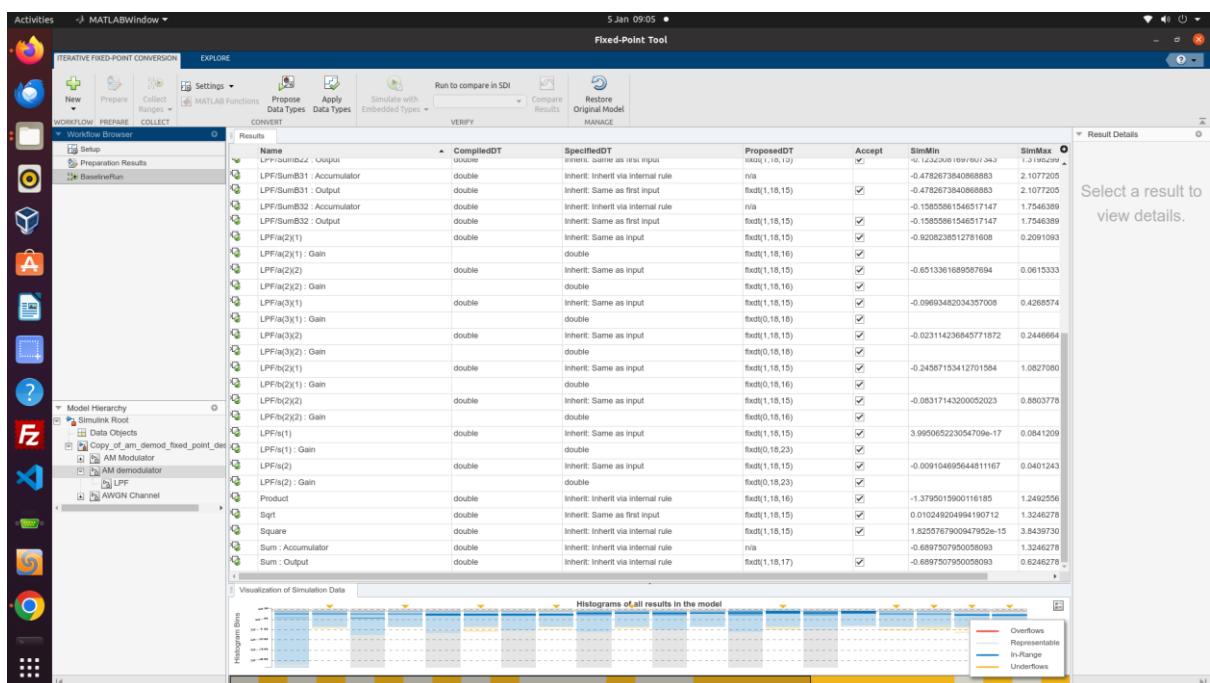
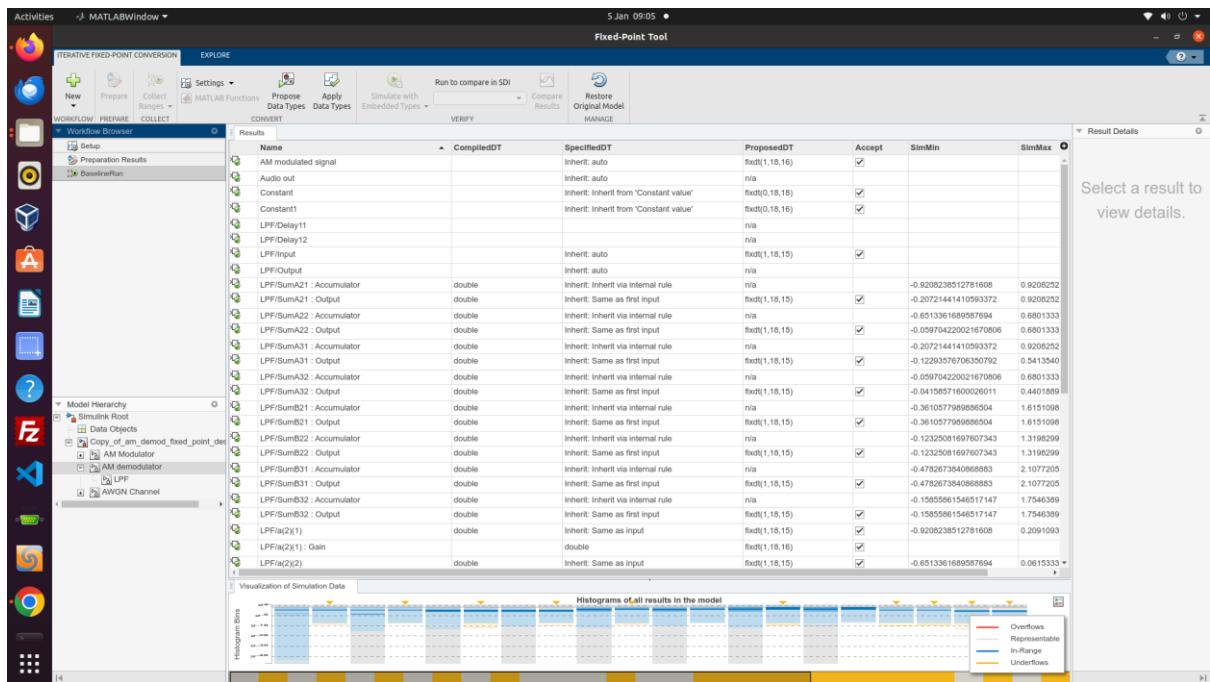
A simulation of the floating point golden reference model gets executed, and data ranges over all the nodes of the block diagram gets collected.

In this way, a dynamic range analysis gets performed.

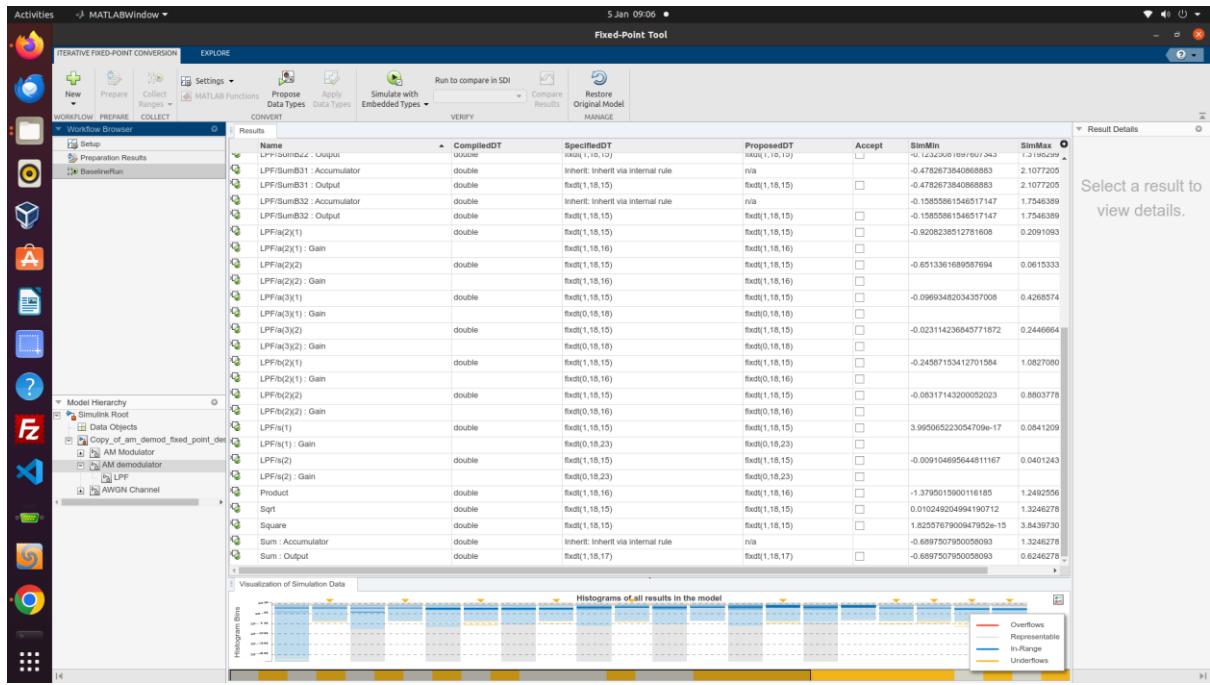
This is used to find out the required fixed point data type representation to use in each point of the block diagram.



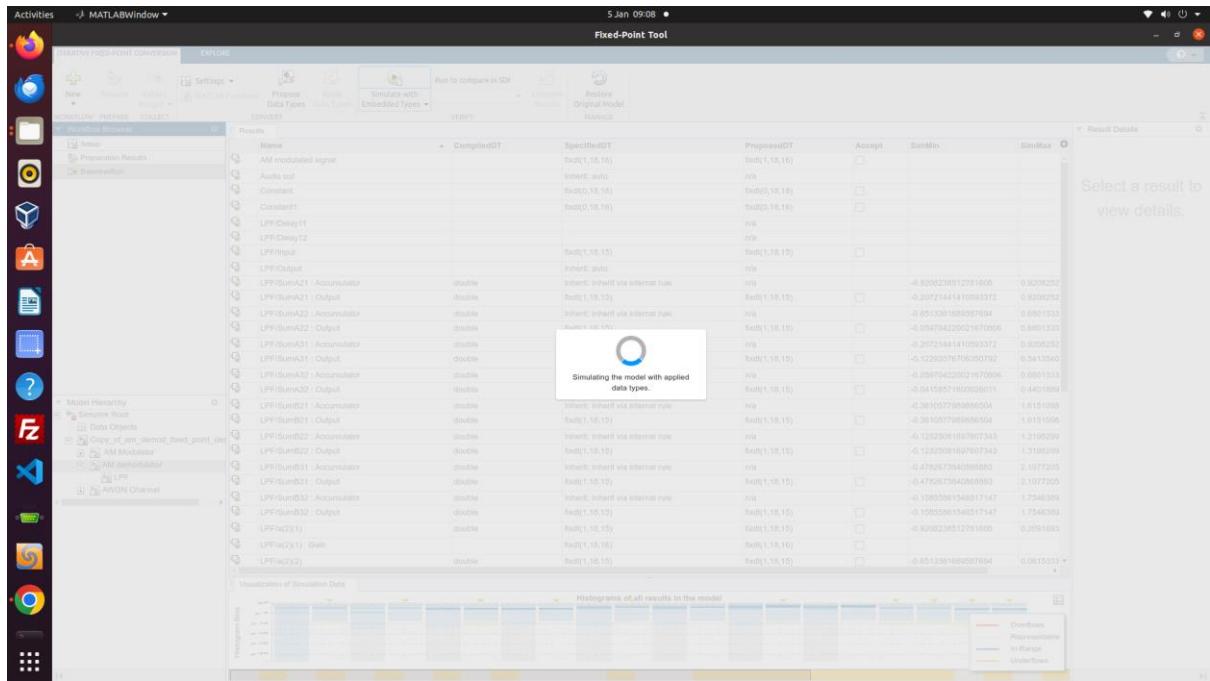
By pressing "propose data types", the automated fixed point data type conversion can be obtained:



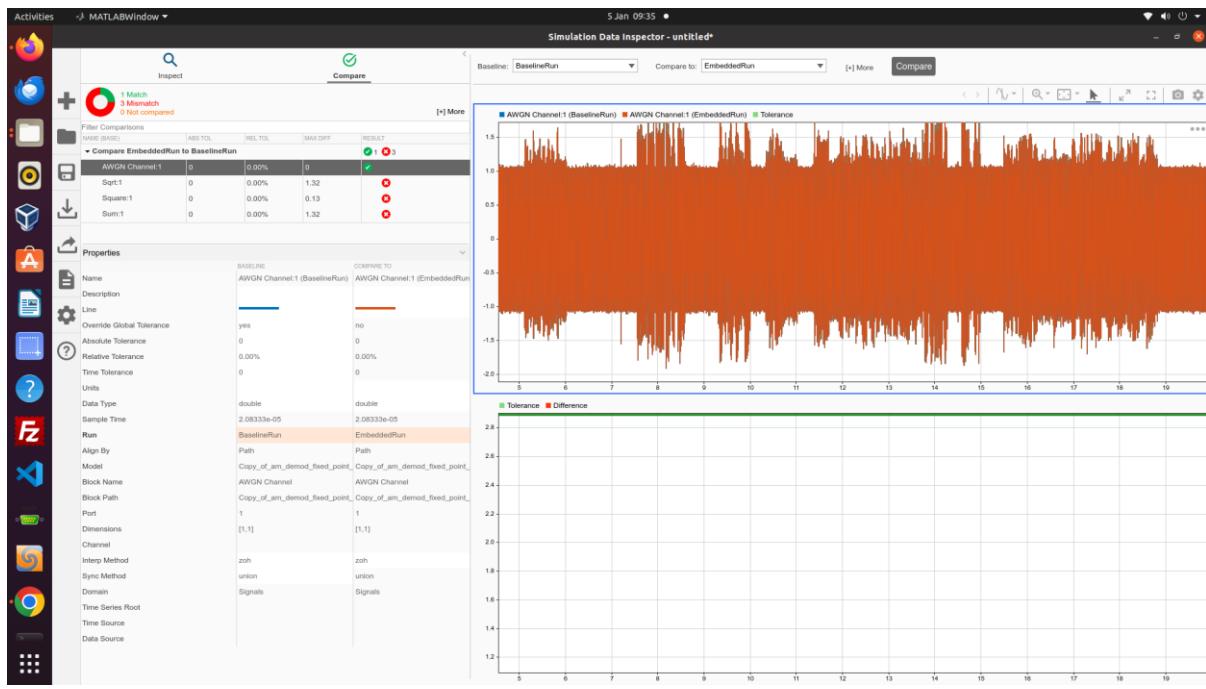
By pressing “apply data types”:



The option “simulate with embedded types” becomes available:

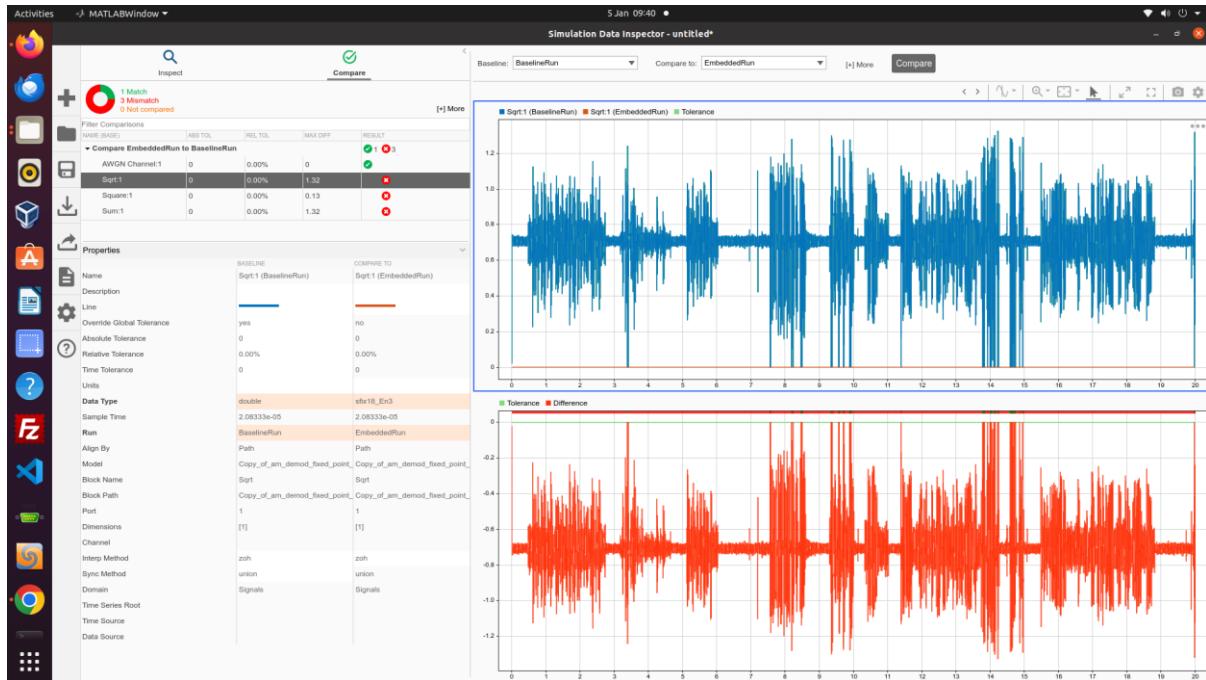


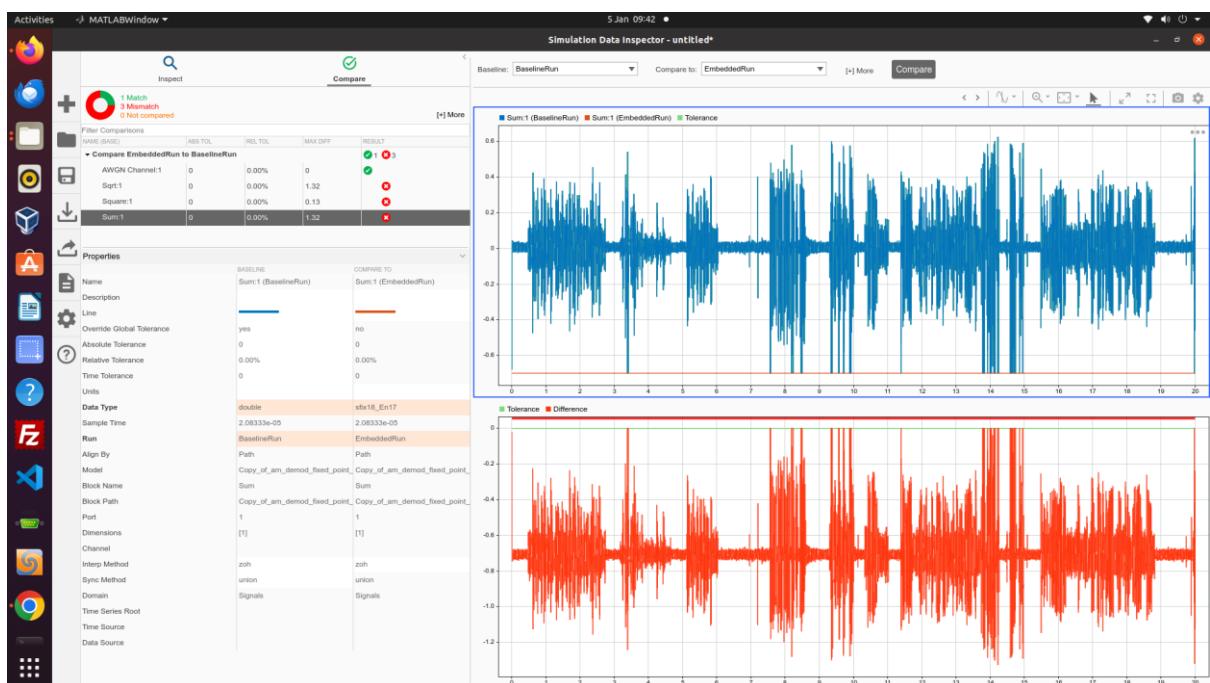
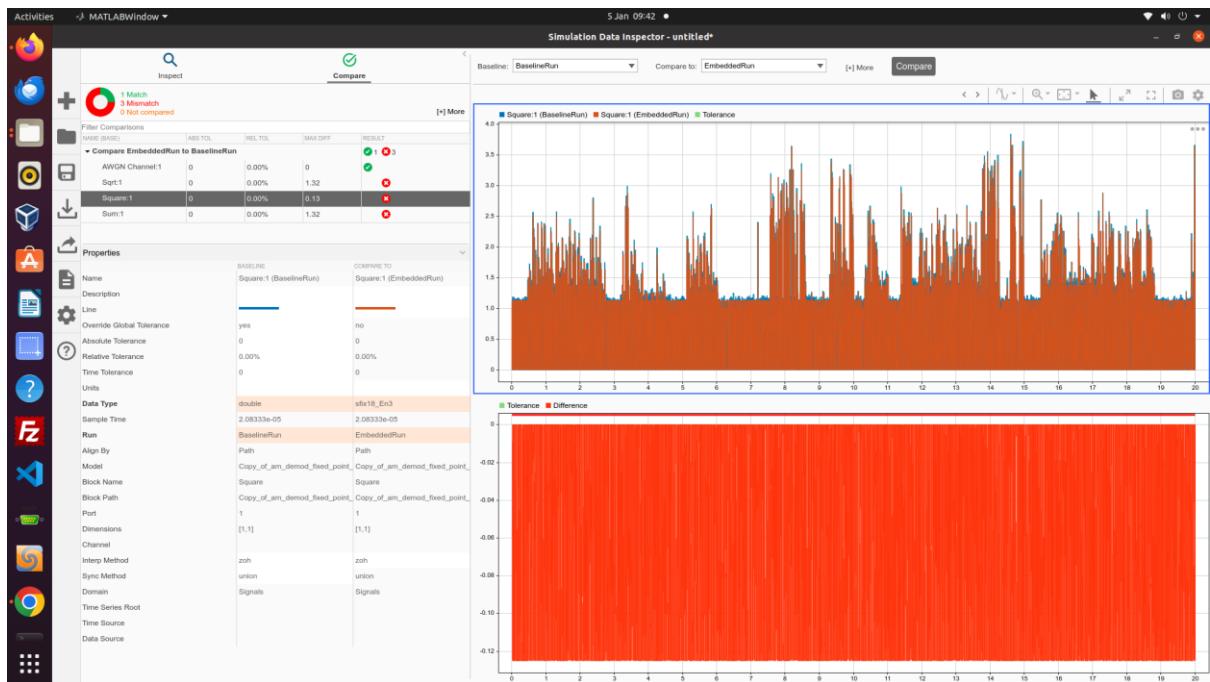
Then a report with simulation outcome of the converted model is obtained:



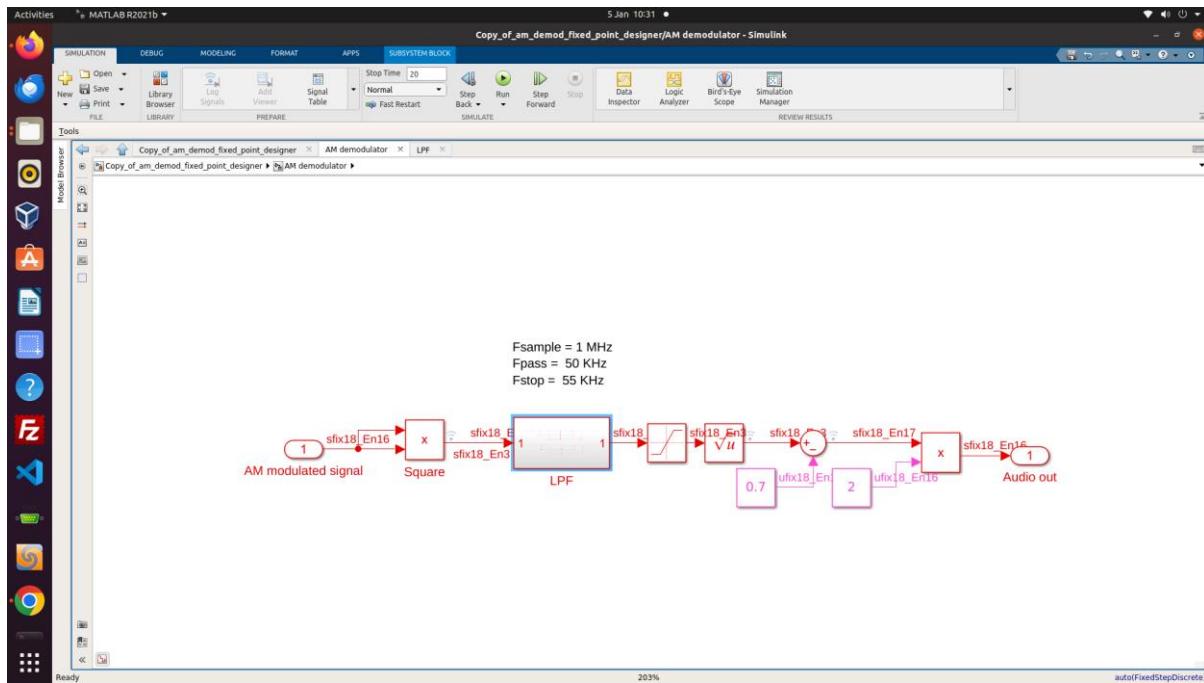
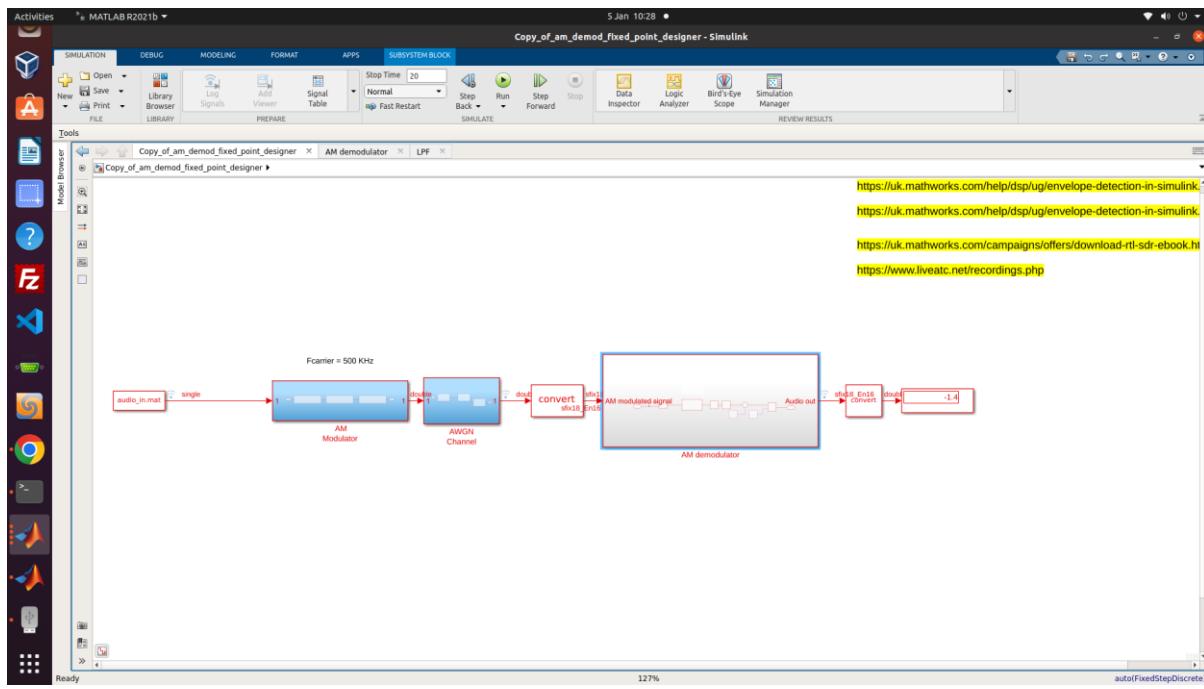
If the differences (if any) are deemed to be acceptable, the conversion finishes.

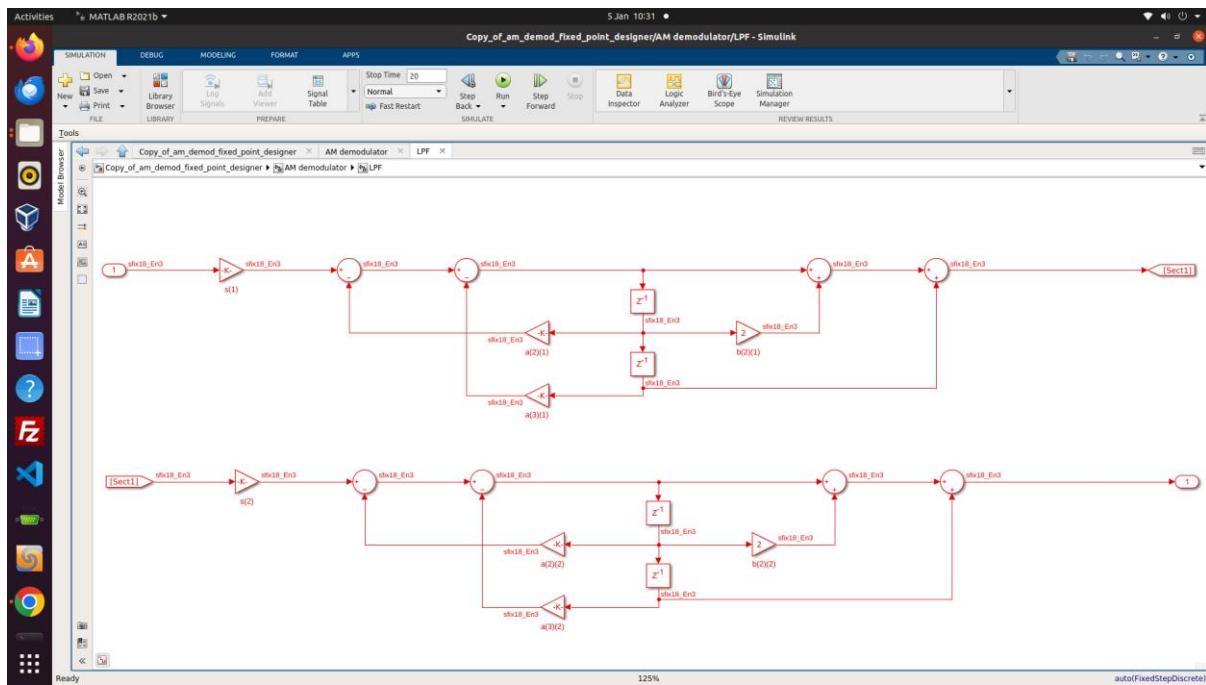
Otherwise, further iterations are possible: by tweaking precisions in nodes in need of further bits, an acceptable end result is obtained over the course of several iterations.



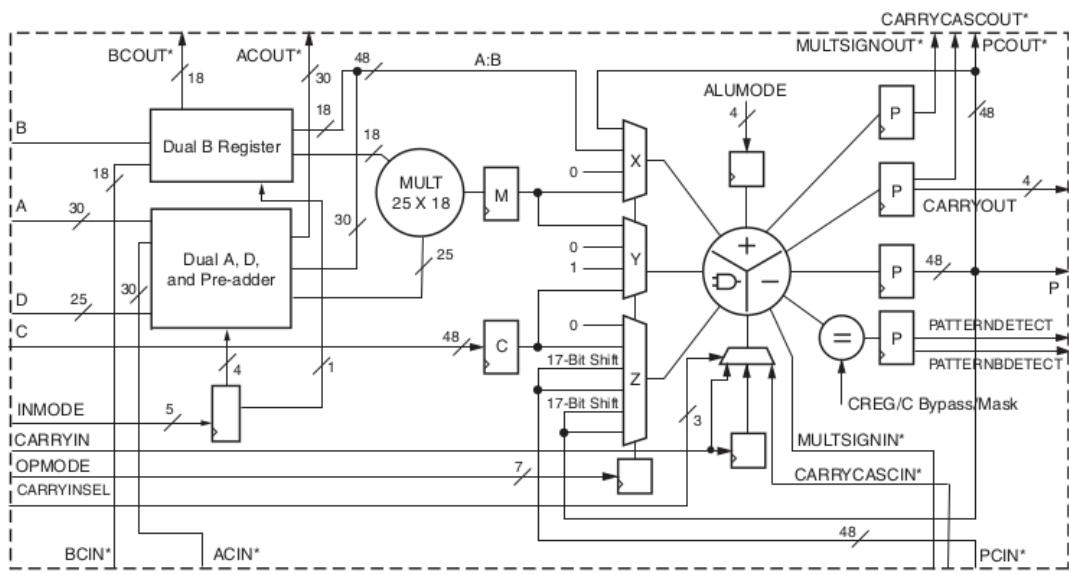


As negative values were reaching the sqrt block, a further saturation block has been added to prevent negative values reaching the block:





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



\*These signals are dedicated routing paths internal to the DSP48E1 column. They are not accessible via fabric routing resources.

UG369\_c1\_01\_052109

Figure 2-1: 7 Series FPGA DSP48E1 Slice

As per UG479:

[https://docs.amd.com/v/u/en-US/ug479\\_7Series\\_DSP48E1](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.

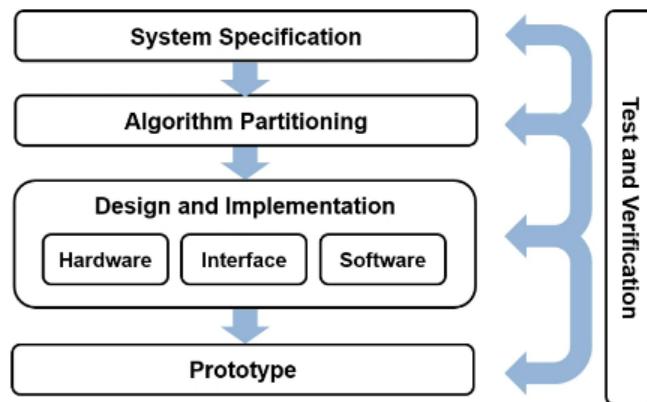
Various algorithms have been developed for effective HW/SW partitioning:

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.

## 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>

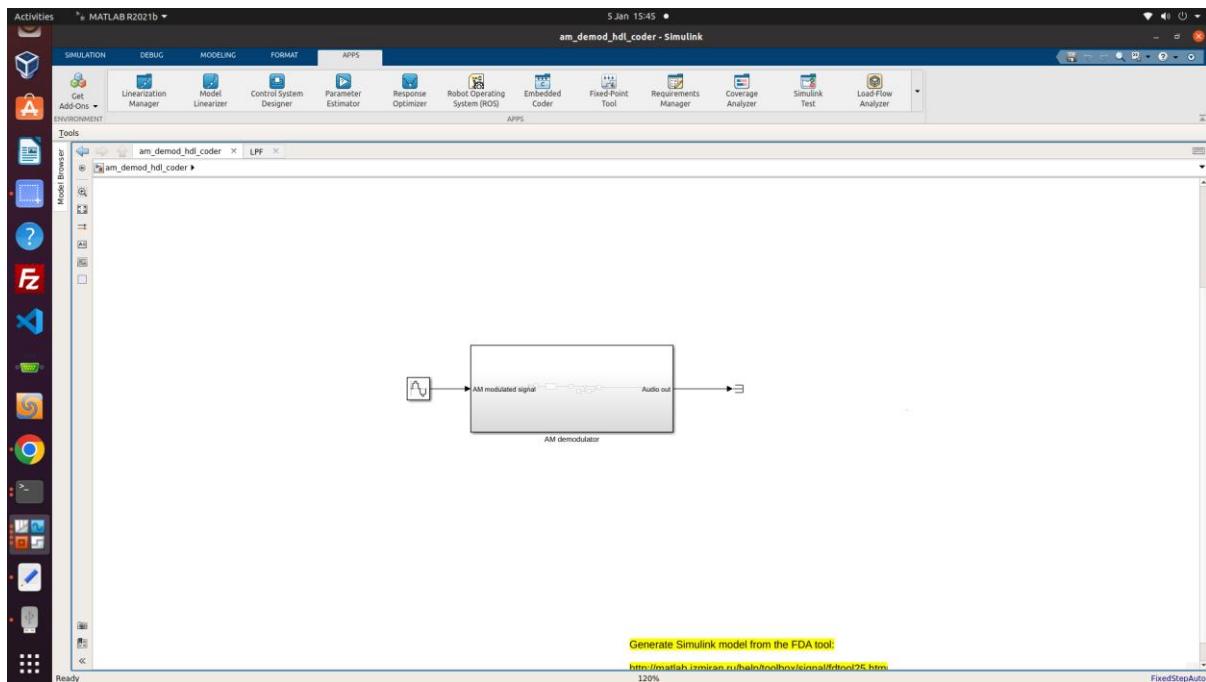
This capability allows engineers to convert algorithmic designs into hardware implementations efficiently.

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.

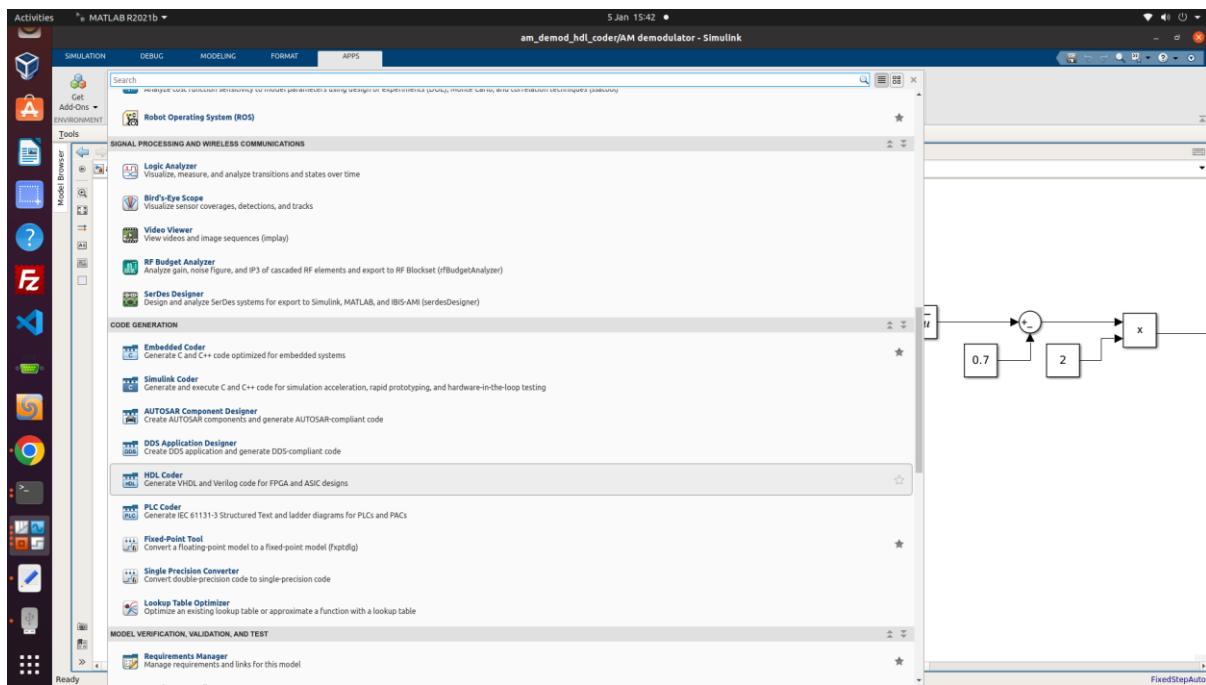
<https://uk.mathworks.com/videos/software-defined-radio-using-matlab-and-simulink-1601496230597.html>

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

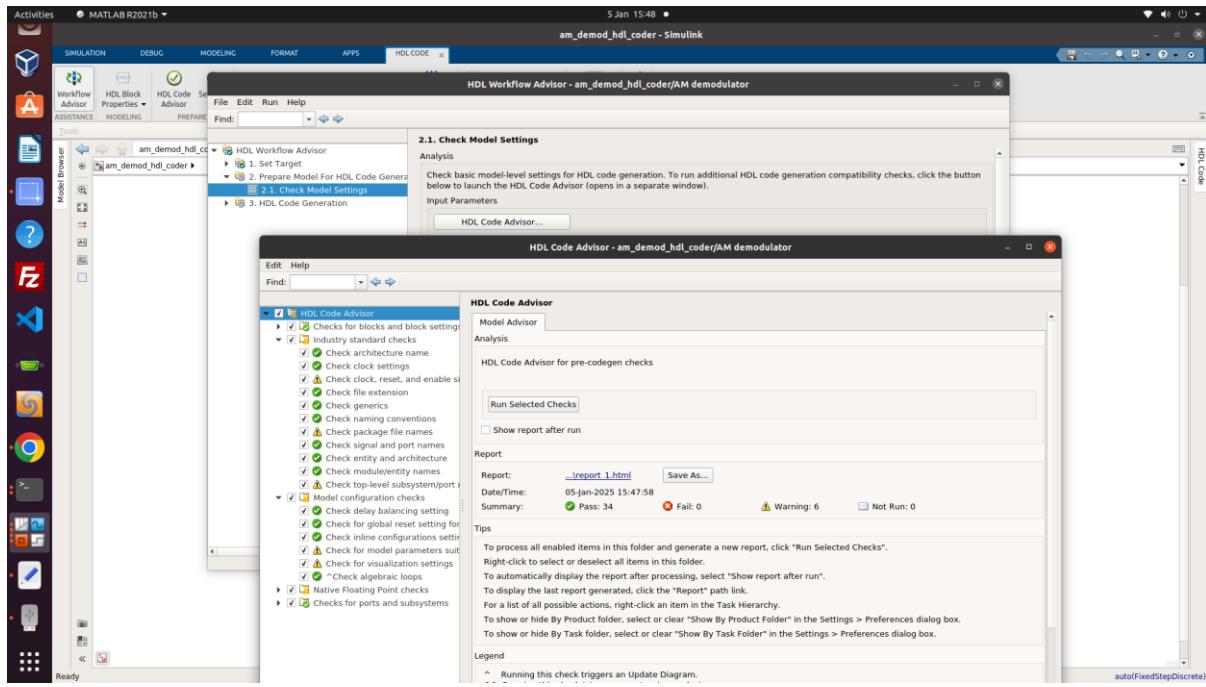


Also in this case audio file blocks have been removed, as being incompatible with the tool.

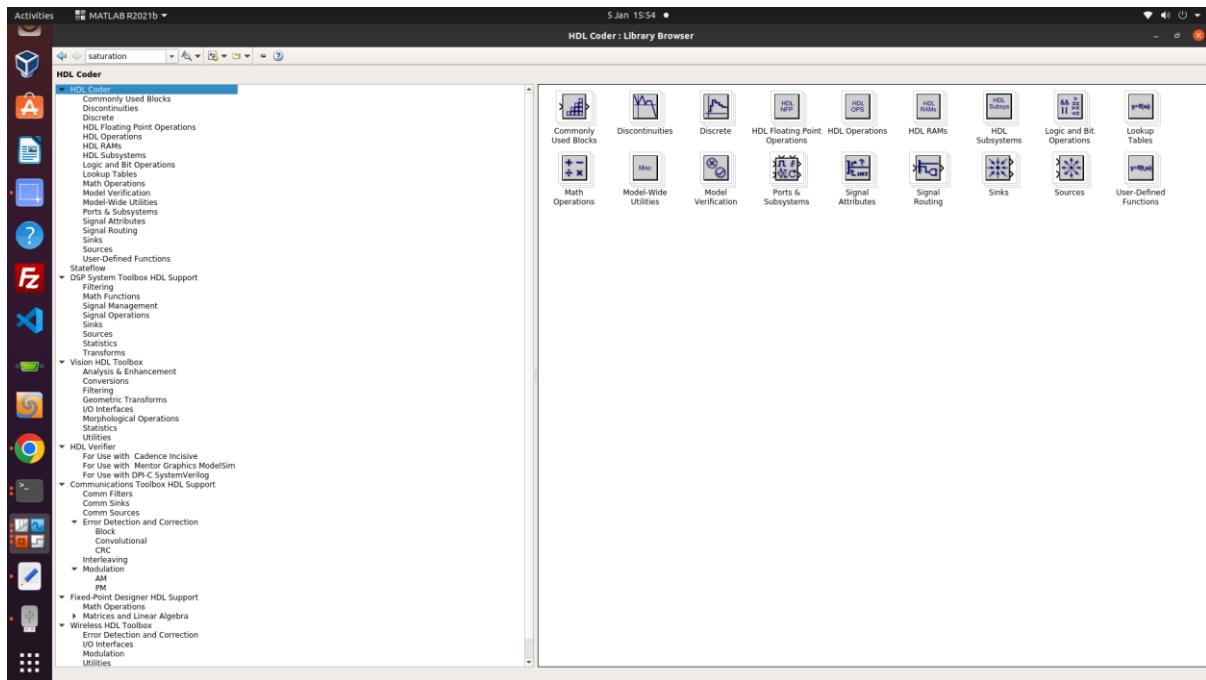
HDL coder gets started as follows:



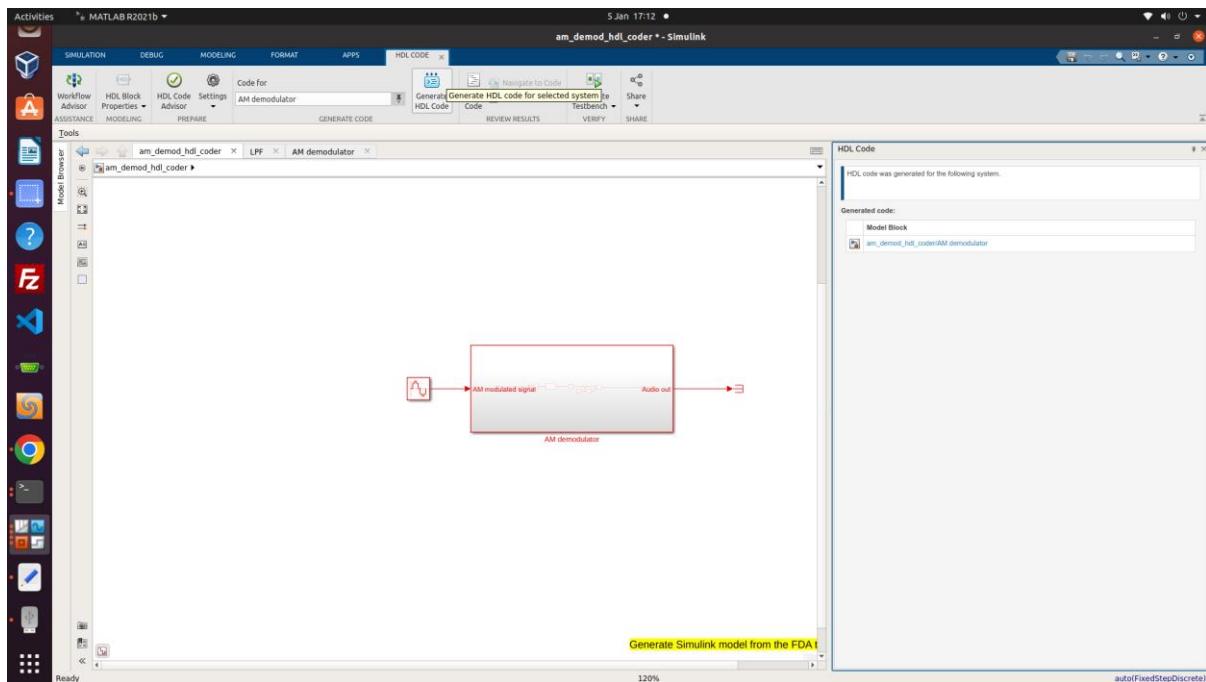
Preliminary checkings on the model can be run via “workflow advisor”:



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

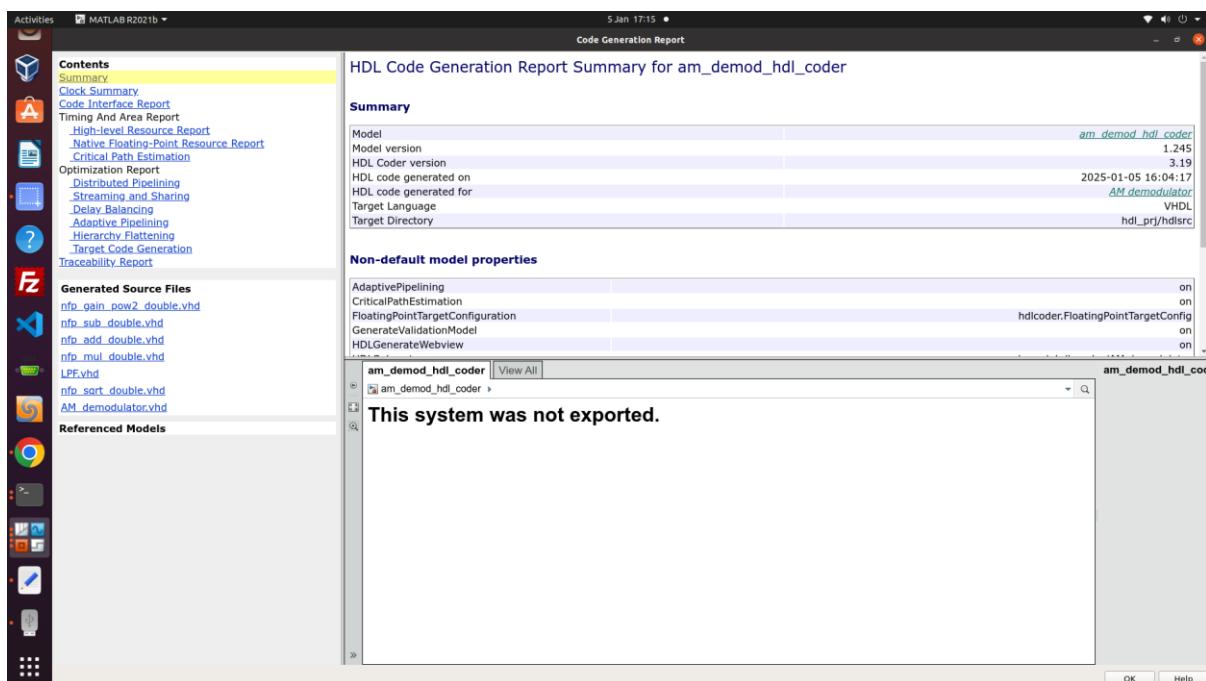


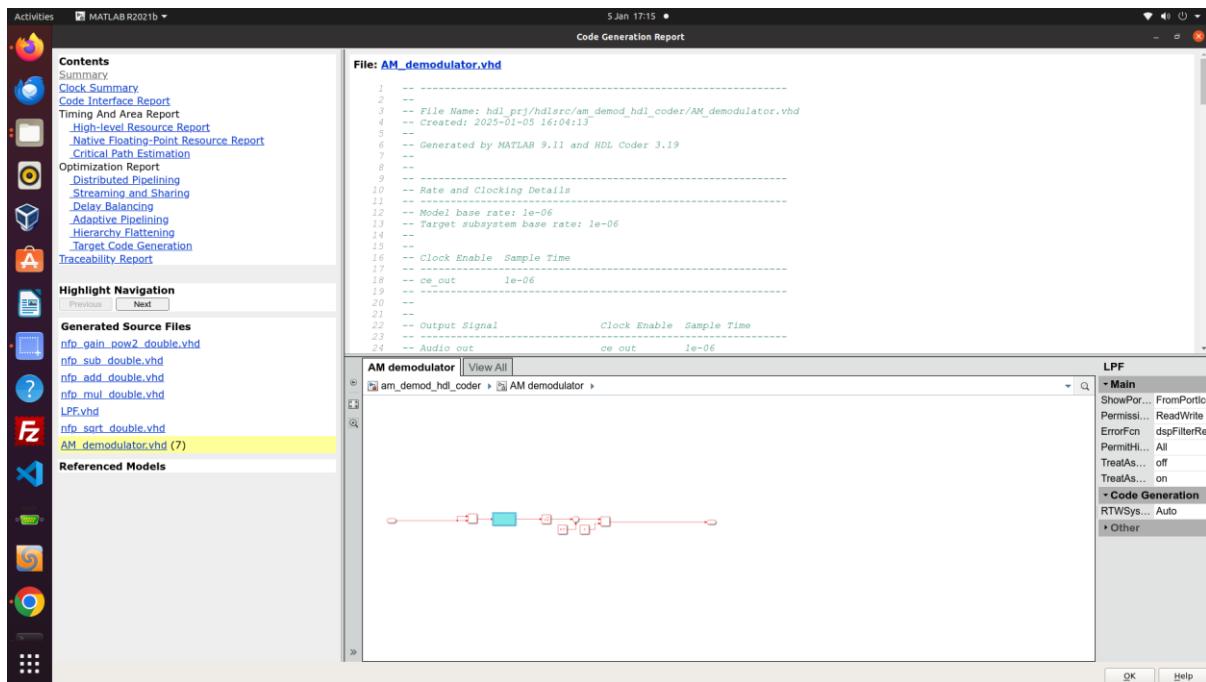
By pressing the generate HDL code:



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 17:16 • 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\_sort\_double.vhd
- AM\_demodulator.vhd

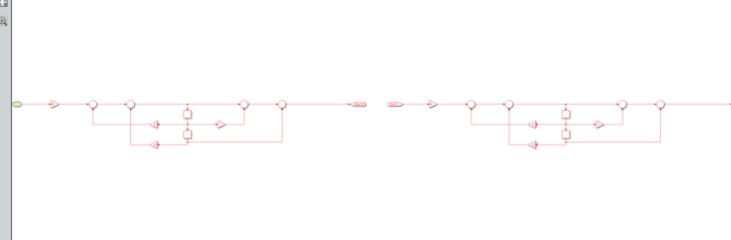
**Referenced Models**

**File: LPF.vhd**

```

1 -- -----
2 --
3 -- File Name: hdl_prj/hdlsrc/am_demod_hdl_coder/LPF.vhd
4 -- Created: 2025-01-05 16:04:13
5 --
6 -- Generated by MATLAB 9.11 and HDL Coder 3.19
7 --
8 --
9 --
10 --
11 -- -----
12 -- Module: LPF
13 -- Source Path: am_demod_hdl_coder/AM demodulator/LPF
14 -- Hierarchy Level: 1
15 --
16 --
17 LIBRARY IEEE;
18 USE IEEE.std_logic_1164.ALL;
19 USE IEEE.numeric_std.ALL;
20
21 ENTITY LPF IS
22 PORT ( clk : IN std_logic;
23         reset : IN std_logic;
24 
```

**LPF** [View All] 



**Input**

**Main**

- Port 1
- IconDisp... Port number
- LatchBy... off
- LatchInp... off

**Signal Attributes**

- OutputP... 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\_sort\_double.vhd
- AM\_demodulator.vhd

**Referenced Models**

**Code Interface Report for am\_demod\_hdl\_coder**

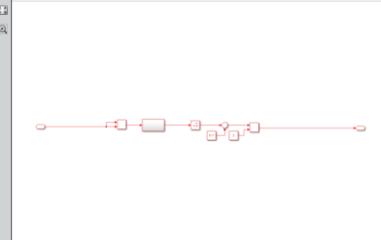
**Input ports**

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 demodulator**

**Main**

- ShowPor... FromPortic
- Permiss... ReadWrite
- ErrorFn...
- PermitHl... All
- TreatAs... off
- TreatAs... on

**Code Generation**

- RTWSys... Auto
- Other

OK Help

Activities MATLAB R2021b • 5 Jan 17:18 •

### Native Floating-Point Resource Report for am\_demod\_hdl\_coder

#### Summary of native floating-point operators

Adders	4
Gain-by-power-of-2	3
Multipilers	7
Sqrt	1
Subtractors	5

#### Detailed Report

##### Module nfp\_add\_double

(Latency = "Zero")

Multipilers	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 demodulator**

AM demodulator [View All] **AM demodulator**

AM demodulator [Main] **Main**  
ShowPort... FromPort  
Permiss... ReadWrite  
ErrorFn  
PermitH... All  
TreatAs... off  
TreatAs... on  
**Code Generation**  
RTWSys... Auto  
Other

OK Help

Activities MATLAB R2021b • 5 Jan 17:19 •

### 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/hdsrc/am\\_demod\\_hdl\\_coder/criticalPathEstimated.m](#)

#### Critical Path Details

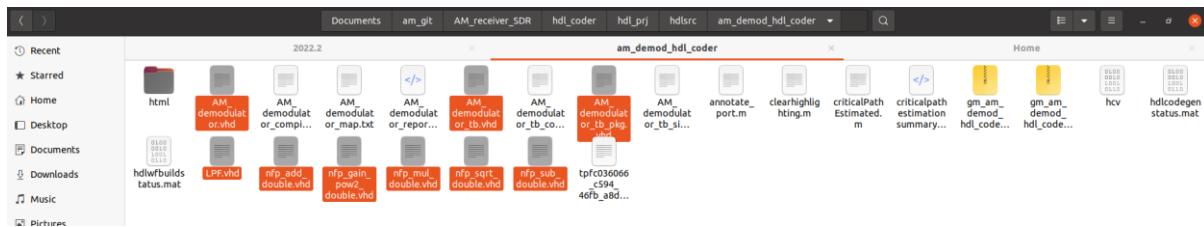
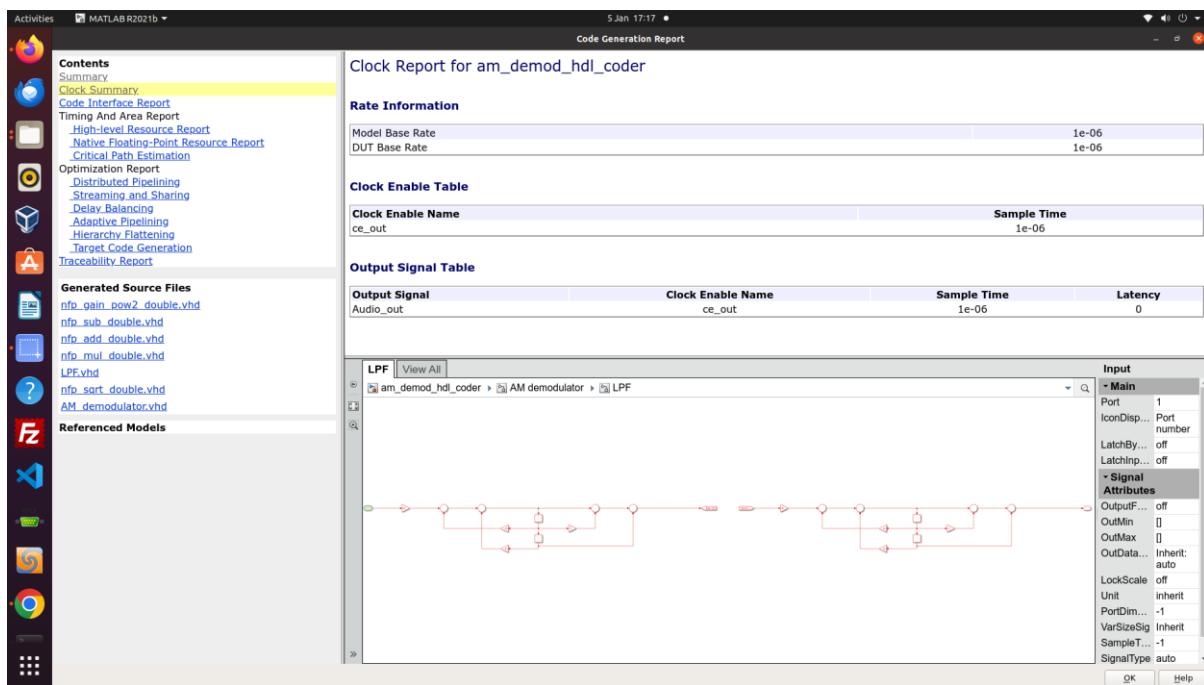
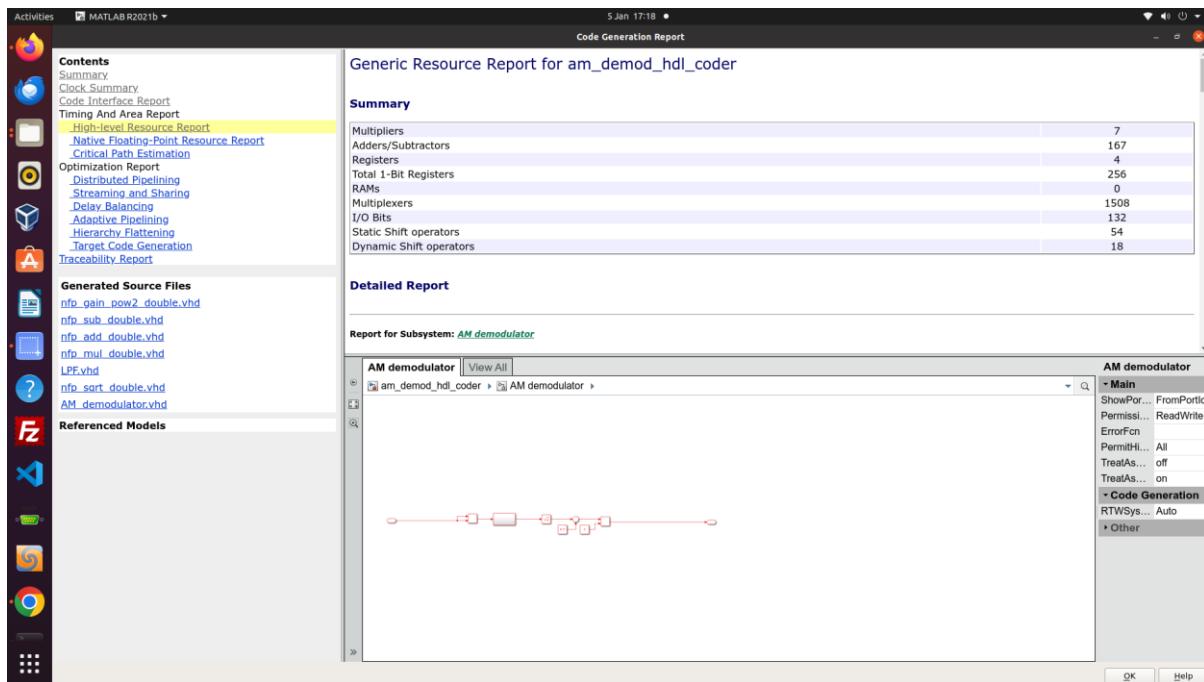
ID	Propagation (ns)	Delay (ns)	Block Path
1	14.8510	14.8510	<i>Square</i>
2	29.6870	14.8360	<i>s(1)</i>
3	47.7300	18.0430	<i>SumA21</i>
4	47.7300	0.0000	<i>SumA31</i>
5	65.7730	18.0430	<i>SumA31_outbuff</i>
6	65.7730	0.0000	<i>SumB21</i>
7	83.8160	18.0430	<i>SumB31</i>
8	101.8590	18.0430	<i>SectCut1</i>
9	101.8590	0.0000	<i>SectIn2</i>
10	101.8590	0.0000	<i>s(2)</i>
11	116.6950	14.8360	<i>SumA22</i>
12	134.7380	18.0430	<i>SumA32_outbuff</i>
13	134.7380	0.0000	<i>SumA32</i>
14	152.7810	18.0430	<i>SumB32</i>
15	152.7810	0.0000	<i>SumR22</i>
16	170.8240	18.0430	<i>Sqrt</i>
17	188.8670	18.0430	<i>Sum</i>
18	287.6790	98.8120	<i>Product_to_gain</i>
19	305.7220	18.0430	
20	307.1910	1.4690	

**AM demodulator** [View All] **AM demodulator**

AM demodulator [View All] **AM demodulator**

AM demodulator [Main] **Main**  
ShowPort... FromPort  
Permiss... ReadWrite  
ErrorFn  
PermitH... All  
TreatAs... off  
TreatAs... on

OK Help



## 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.

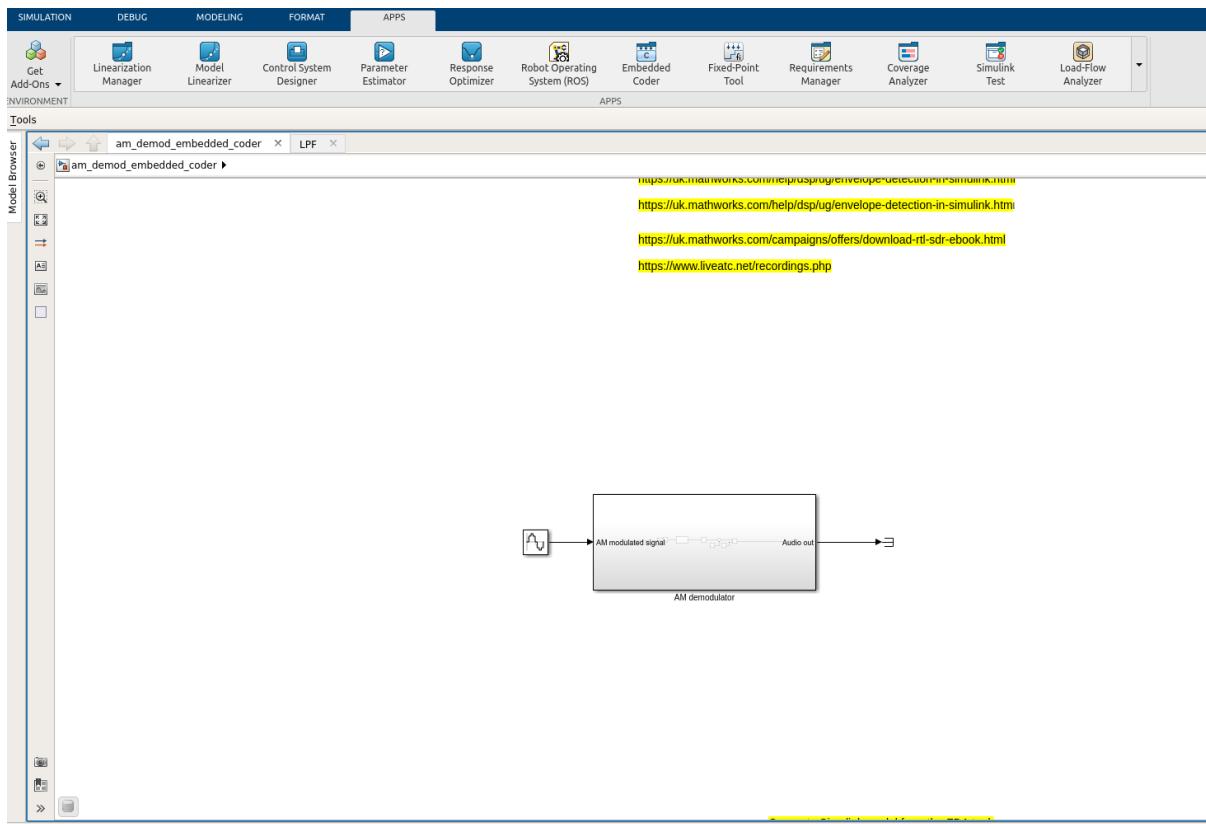
<https://www.mathworks.com/products/embedded-coder.html>

Its key features are:

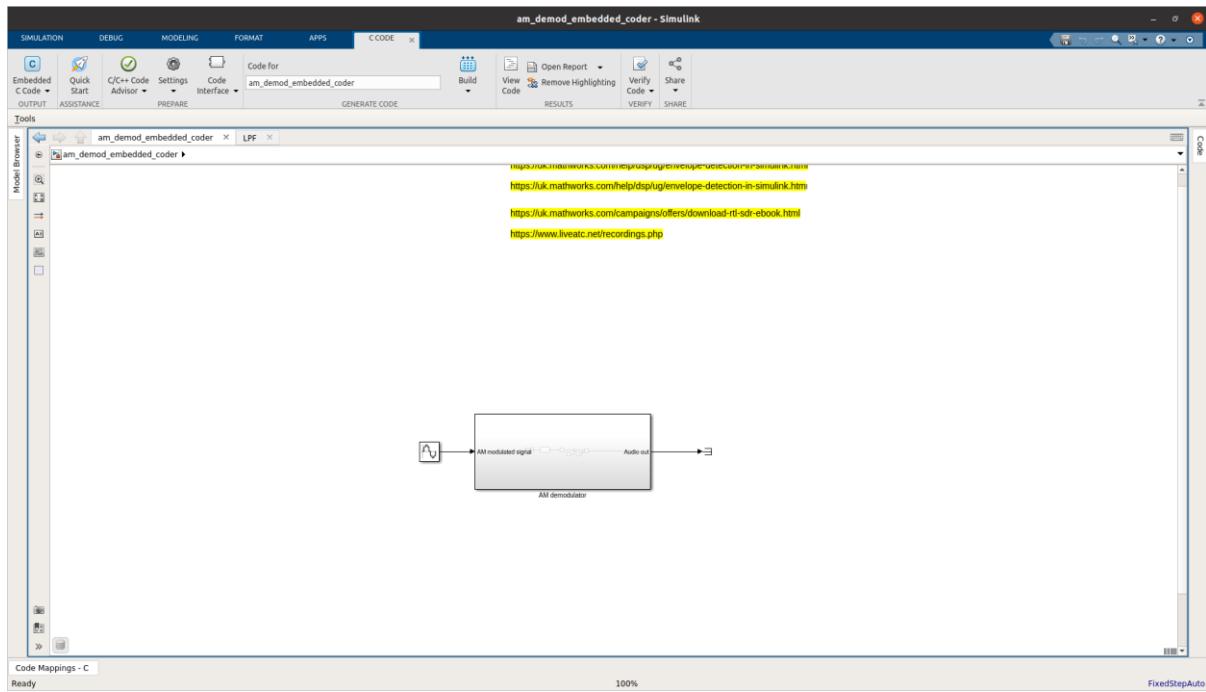
- Code Generation: Embedded Coder generates optimized code from models created in MATLAB and Simulink. This code can be tailored to meet specific requirements for embedded systems, ensuring that it is both efficient and portable.
- Optimization Techniques: The product incorporates various optimization 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.

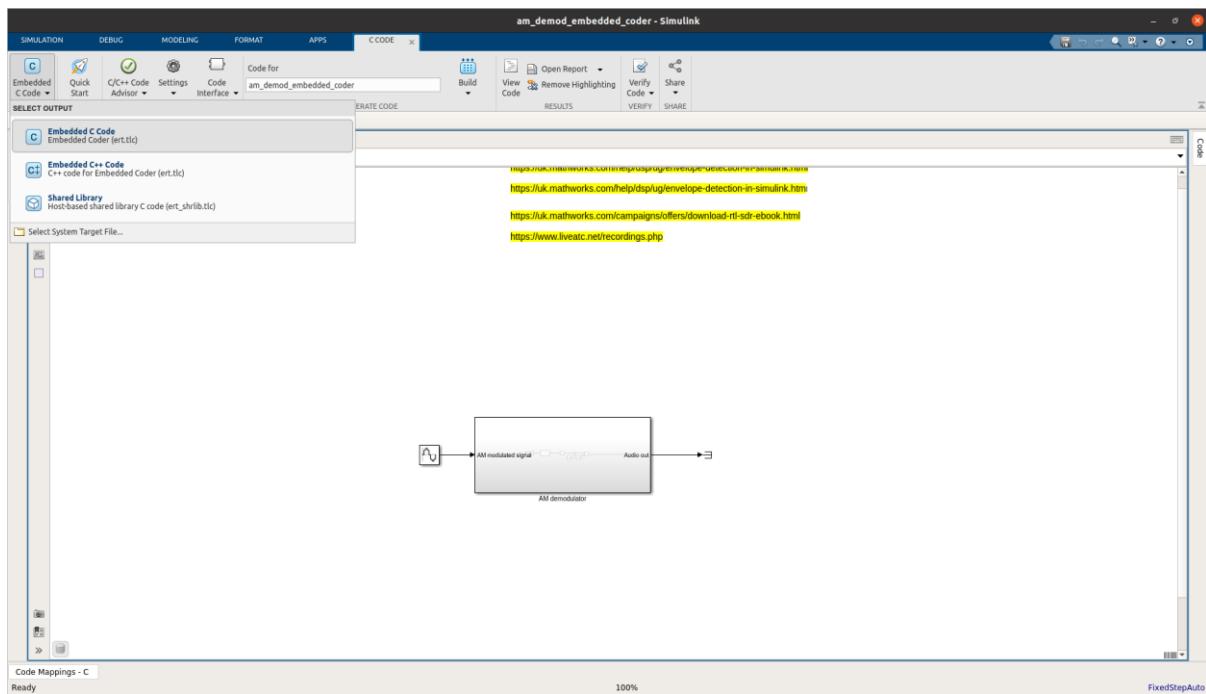
The starting model is:



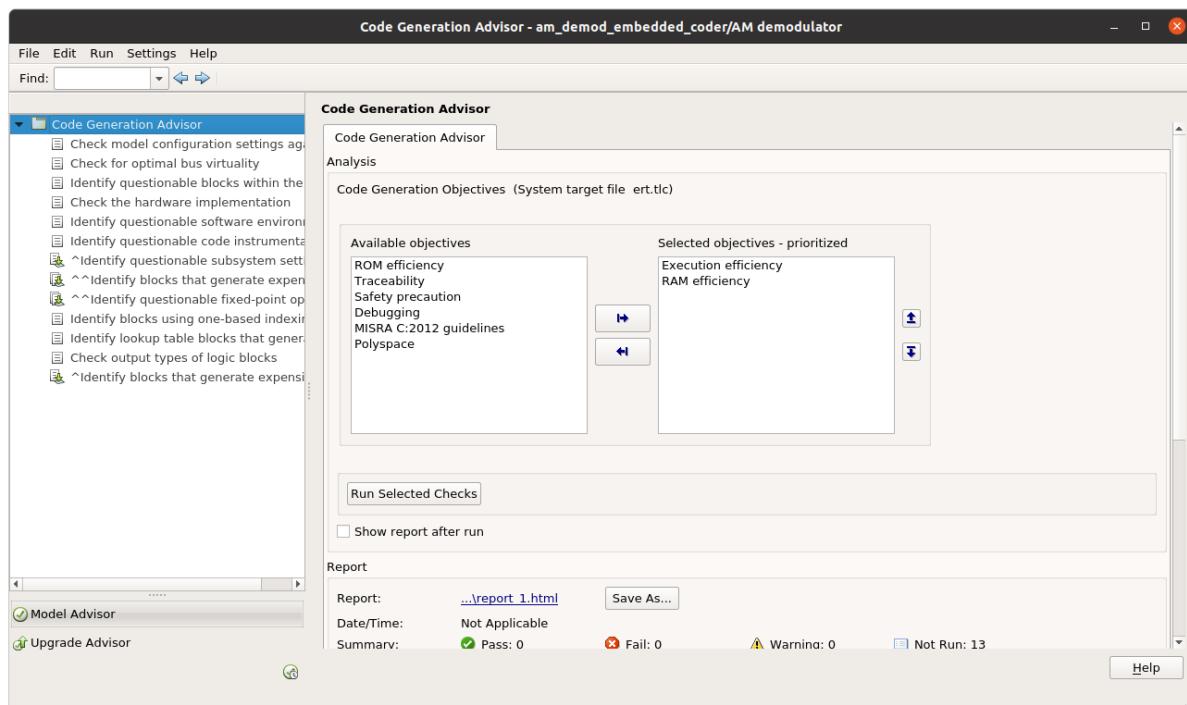
Invoking the embedded coder app:



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 for 'am\_demod\_embedded\_coder'**

### Model Information

Author	caccolillo
Last Modified By	caccolillo
Model Version	1.247
Tasking Mode	MultiTasking

**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 Interface Report for am\_demod\_embedded\_coder**

### Table of Contents

- Entry-Point Functions
- Imports
- Outputs
- Interface Parameters
- Data Stores

#### Entry-Point Functions

Function: [am\\_demod\\_embedded\\_coder\\_initialize](#)

Prototype	<b>void am_demod_embedded_coder_initialize(void)</b>
Description	Initialization entry point of generated code
Timing	Must be called exactly once
Arguments	None
Return value	None
Header file	<a href="#">am_demod_embedded_coder.h</a>

Function: [am\\_demod\\_embedded\\_coder\\_step](#)

Prototype	<b>void am_demod_embedded_coder_step(void)</b>
Description	Output entry point of generated code
Timing	Must be called periodically, every 1e-06 seconds
Arguments	None
Return value	None
Header file	<a href="#">am_demod_embedded_coder.h</a>

Function: [am\\_demod\\_embedded\\_coder\\_terminate](#)

Prototype	<b>void am_demod_embedded_coder_terminate(void)</b>
Description	Termination entry point of generated code
Timing	Must be called exactly once
Arguments	None
Return value	None
Header file	<a href="#">am_demod_embedded_coder.h</a>

#### Imports

All functions in model

Code Generation Report

File /home/caccilillo/Documents/am\_git/AM\_receiver\_SDR/embedded\_coder/generated\_code/model\_composer\_am\_demod\_v6/am\_demod\_embedded\_coder\_ert\_rtw/html/index.html

Content

am\_demod\_embedded\_coder.c

```
/* File: am_demod_embedded_coder.c
 * Generated: 2023-05-15 13:47:44
 * Model: am Demod Embedded Coder
 * Version: 1.347
 * Simulink Version: 9.8 (R2023a) 14-May-2023
 * C/C++ source code generated on: Wed May 15 13:47:44 2023
 *
 * Target selection: armcortexm
 * Embedded hardware selection: ARM Compatible->ARM Cortex
 * Configuration: Configuration is unspecified
 * Validation result: Not run
 */
#include "am_demod_embedded_coder.h"
#include "am_demod_embedded_coder_private.h"

/* Block states (default storage) */
DM_am_demod_embedded_coder_T am_demod_embedded_coder_DM;

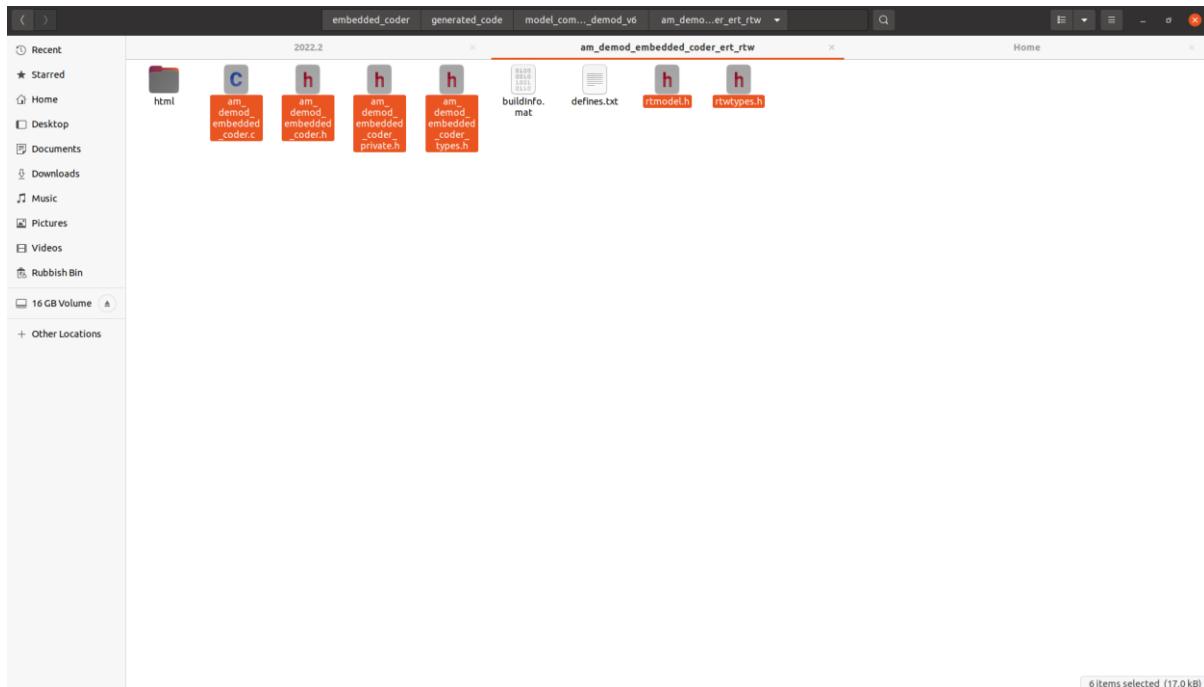
/* Real-time model */
static RT_MODEL_am_demod_embedded_coder_T am_demod_embedded_coder_M;
RT_MODEL_am_demod_embedded_coder_T amDemodEmbeddedCoder_M =
    am_Demod_Embdedded_Coder_M;

/* Model step function */
void am_demod_embedded_coder_step(void)
{
    real_T lastIn_tm;
    real_T rtwDelay_tm;
    real_T rtwProduct;
    real_T rtwSqrt;
    real_T sum_tm;

    /* Sini: <root>/sim/Wave */
    if (am_demod_embedded_coder_DM.systemenable != 0) {
        lastIn_tm = amDemodEmbeddedCoder_M.Timing.clockTick;
        amDemodEmbeddedCoder_M.Timing.clockTick = (amDemodEmbeddedCoder_M.Timing.clockTick * 4294967296.0) + 1.0E-4;
        amDemodEmbeddedCoder_M.lastIn = sin(lastIn_tm);
        amDemodEmbeddedCoder_M.lastOut = constlastIn_tm;
        amDemodEmbeddedCoder_DM.systemable = 0;
    }

    lastIn_tm = amDemodEmbeddedCoder_DM.lastIn * 0.9999999999999995;
    rtwSqrt = (amDemodEmbeddedCoder_DM.lastIn * (-0.9999999999999993E-7)) +
        lastIn_tm * 0.9999999999999995;
    amDemodEmbeddedCoder_DM.lastOut = (amDemodEmbeddedCoder_DM.lastIn * sin(lastIn_tm)) +
        9.999999999999993E-7) * 1.0 + 0.8;

    /* End of Sini: <root>/sim/Wave */
    /* Sum: <G1>/SumG1' incorporates:
     * Delay: <G1>/Delay1'
     * Delay: <G1>/Delay2'
     * Gain: <G1>/Gain1'
     * Gain: <G1>/Gain2'
     * Sum: <G1>/SumG1'
     */
    rtwSqrt = rtwSqrt * rtwSqrt * 0.82148031067940924 - 1.0;
    amDemodEmbeddedCoder_M.lastOut = rtwSqrt;
}
```



# 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.

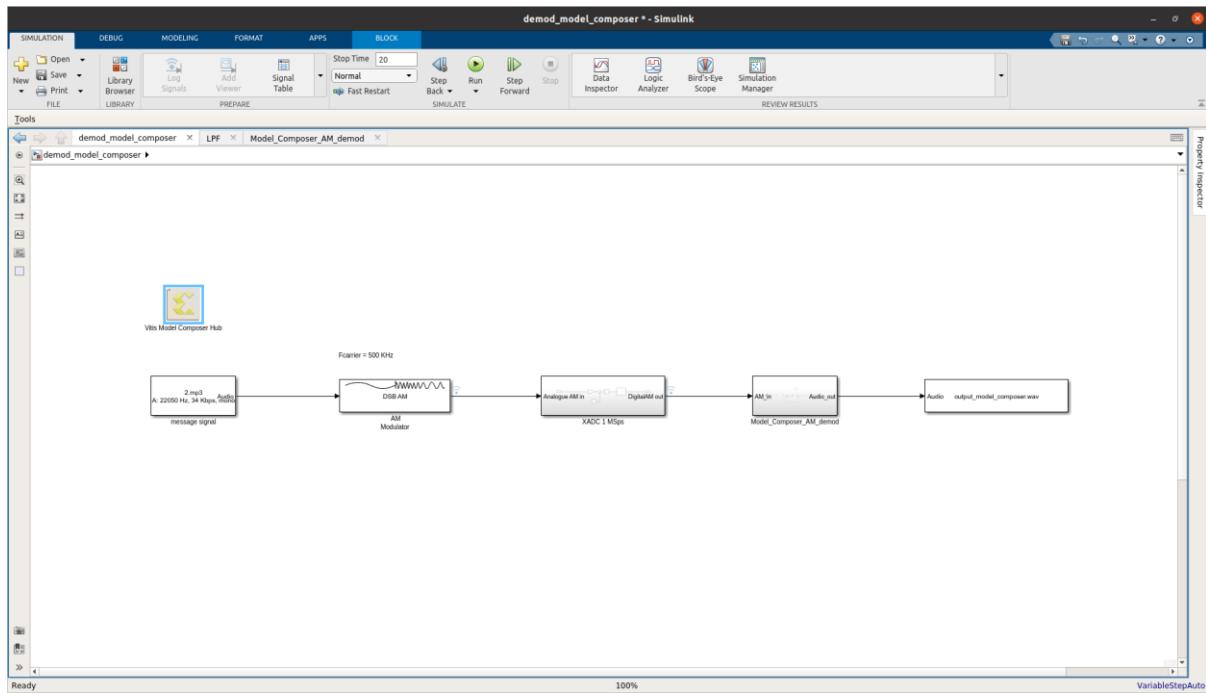
Users can import custom High-Level Synthesis (HLS), AI Engine, and Register Transfer Level (RTL) code as Intellectual Property (IP) blocks into their designs, enhancing flexibility and reusability.

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>

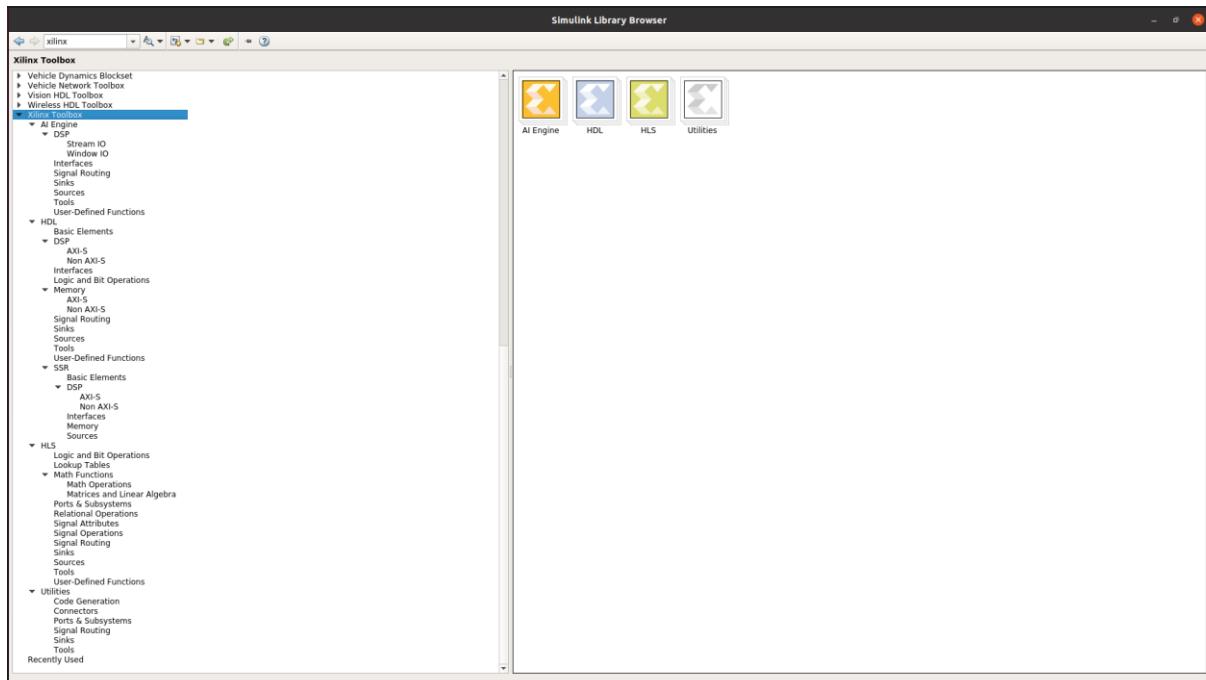
The model made by using model composer is:

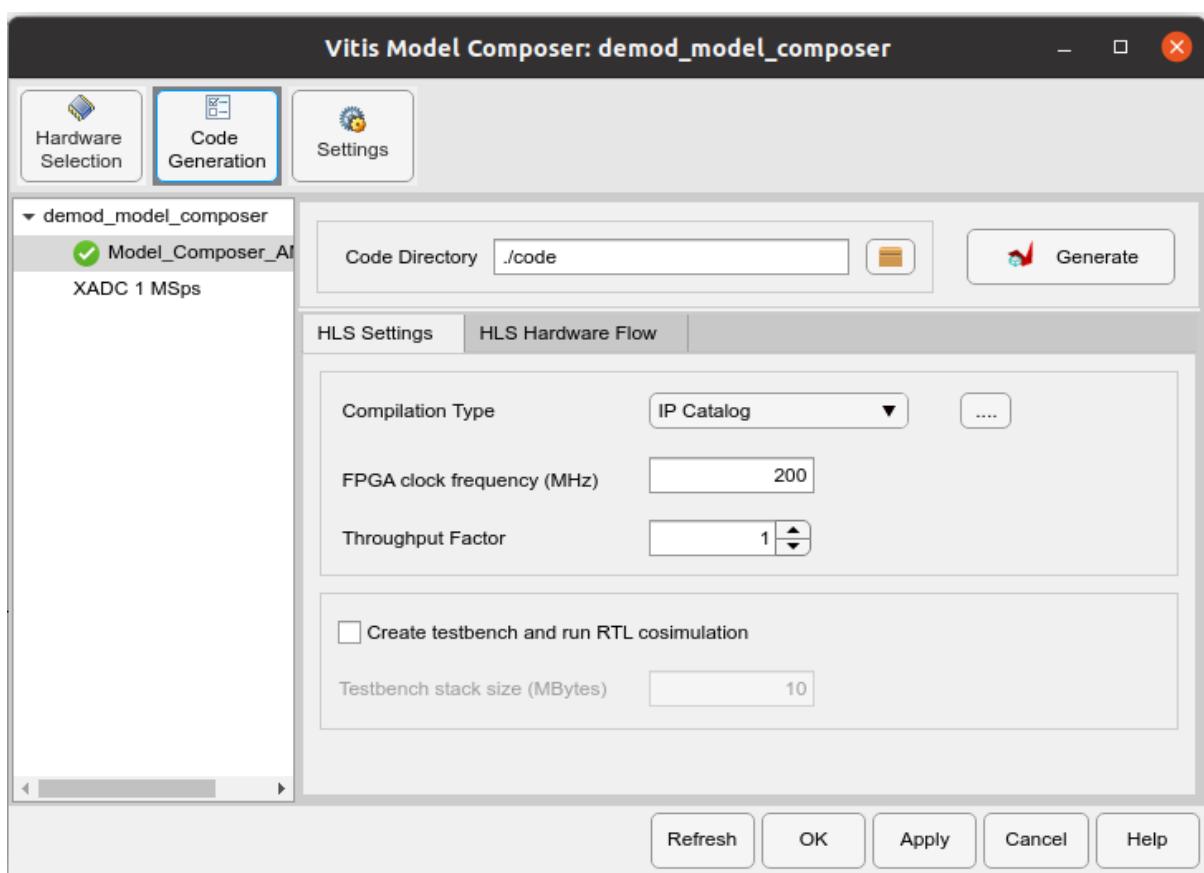
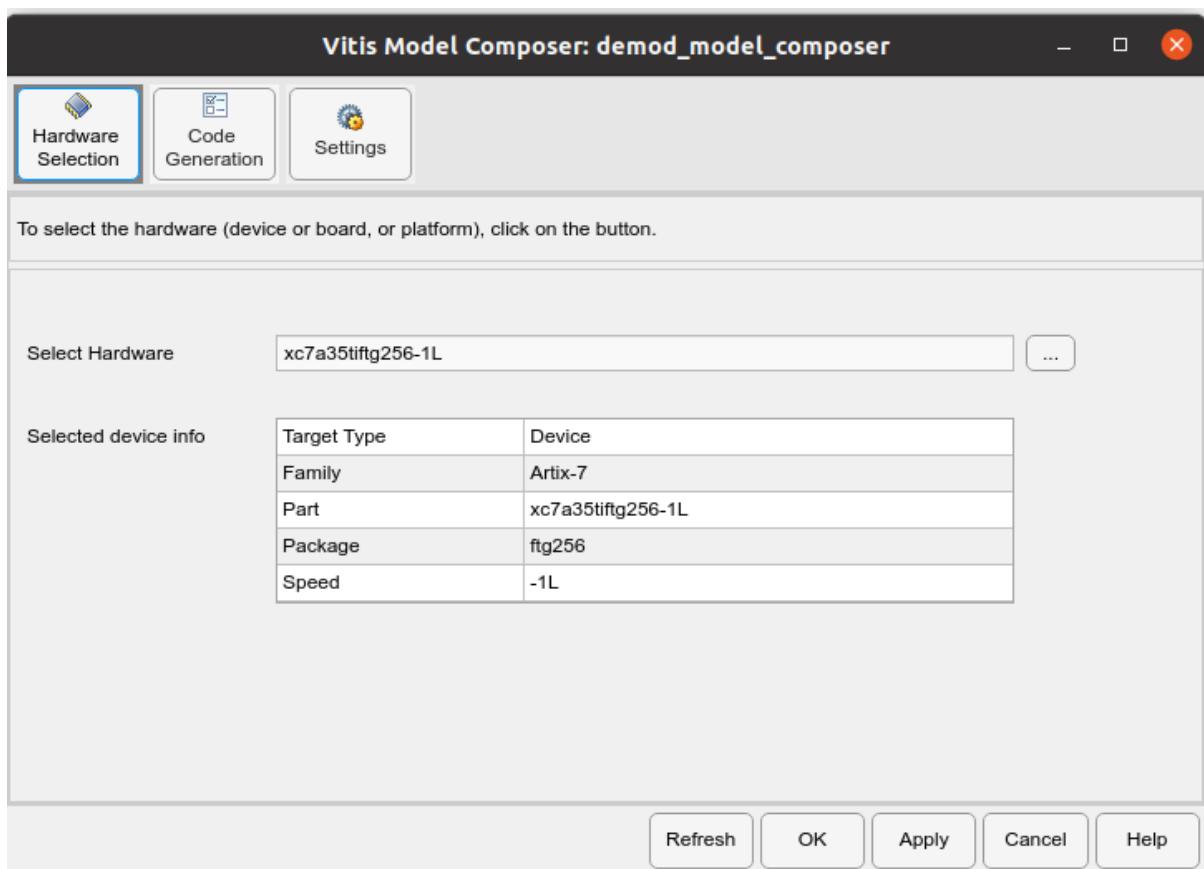


Being a tool suited for Xilinx devices only, it stands at a little bit lower abstraction layer than HDL coder.

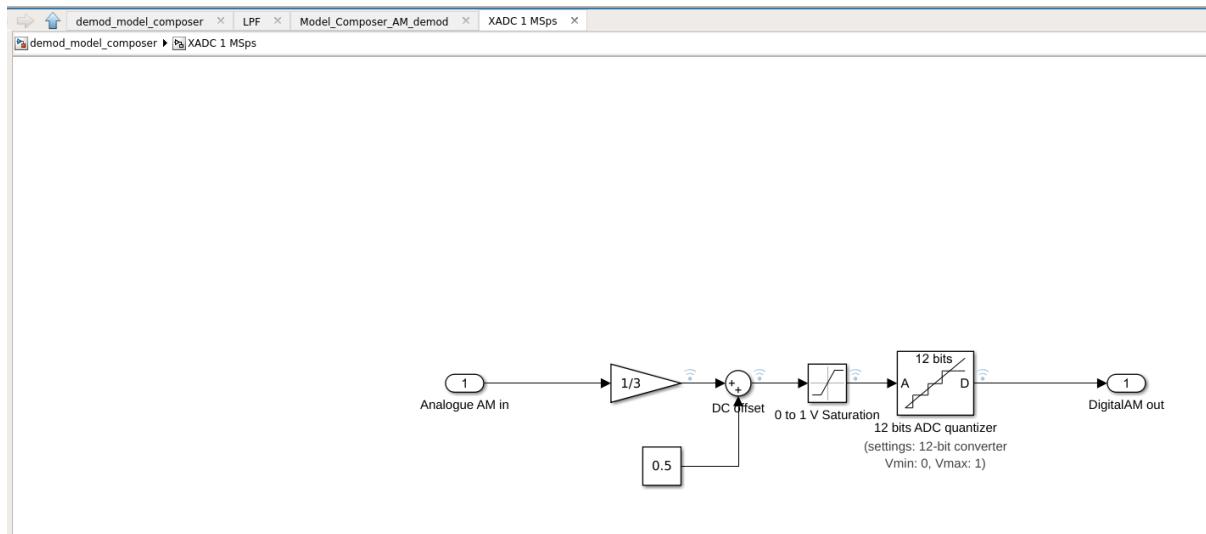
In the library of available blocks, there's plenty of Xilinx specific low level primitives:

<https://www.youtube.com/watch?v=nMfYF4daxal>

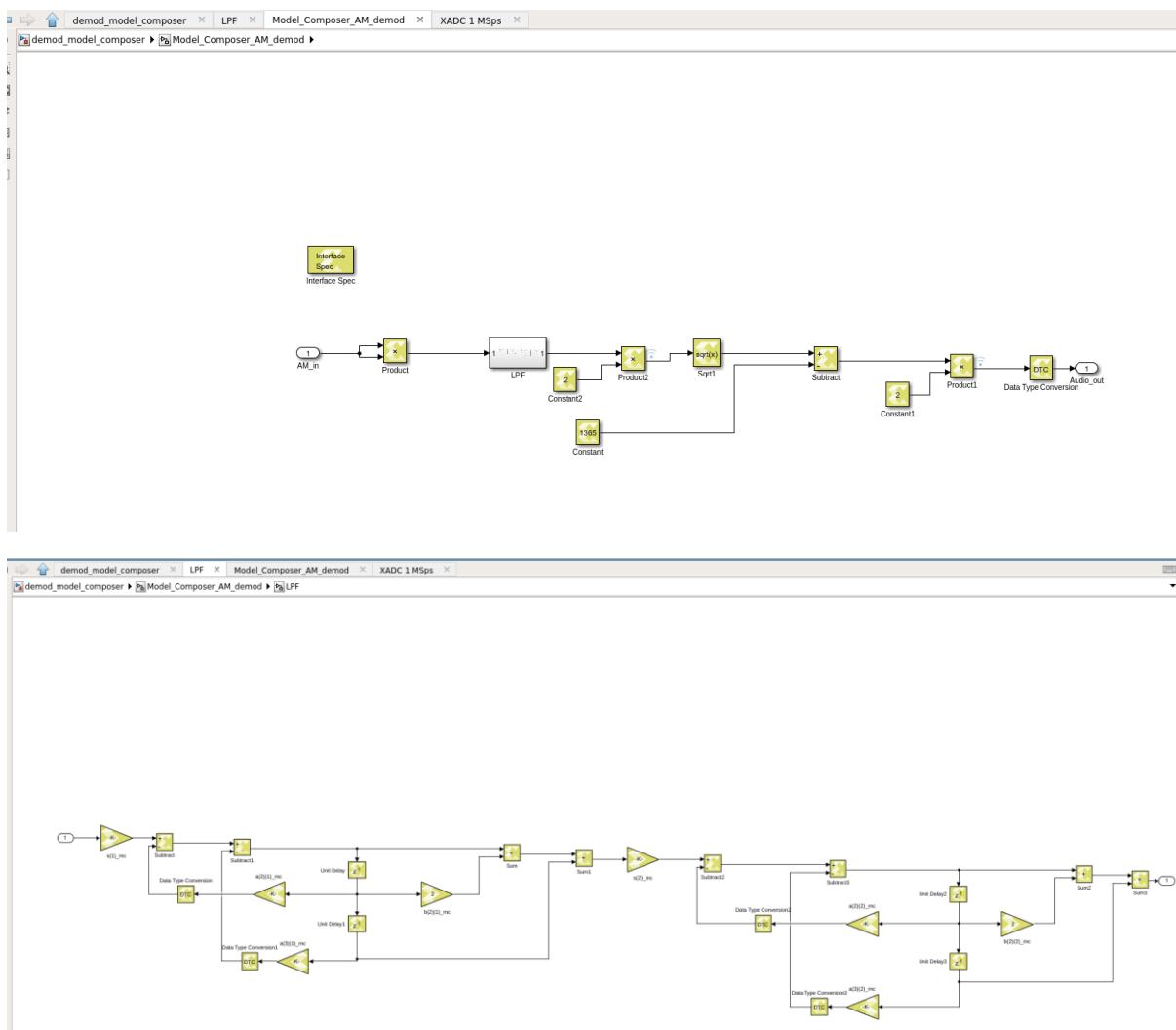




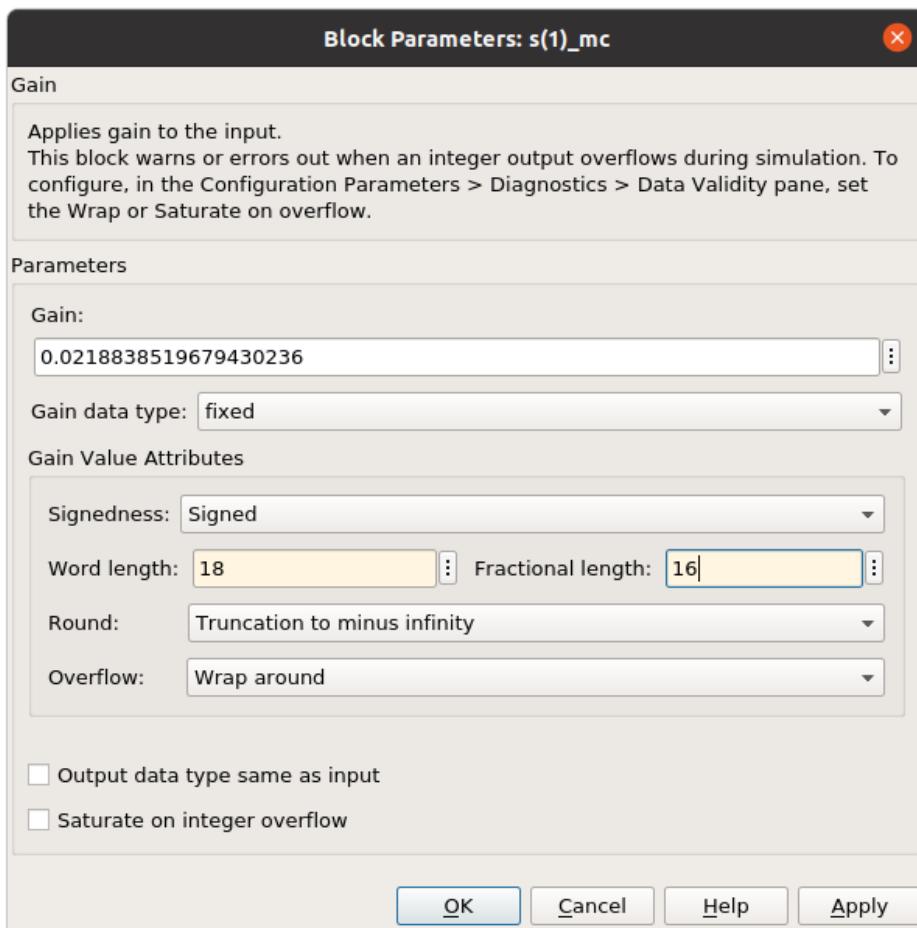
In this model, a rough modelling of the effects of the XADC have been introduced:



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](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>

## Setting up HIL Test System:



Simon's Team  
HIL Systems Engineering

Powered  
actuators



Electronic  
loads



Power  
Supplies

