

MATLAB™ HDL Coder

Introducción al Toolbox de MATLAB™

Dr. Pablo COSSUTTA

Julio 2020

- Verilog/VHDL son herramientas que simplifican el diseño de circuitos digitales
 - ▶ No fueron pensadas para realizar algoritmos complejos
 - ▶ La aritmética de Punto Fijo es complicada
 - Ya que el usuario/programador debe corregir el punto acordemente en cada operación
 - ▶ La aritmética de Punto Flotante es costosa
 - Utiliza una gran cantidad de recursos
 - Alta latencia (generalmente 12 ciclos por operación)
- Existen diversas herramientas
 - ▶ Librerías para utilizar en lenguajes HDL
 - Sigue siendo necesario escribir código complejo
 - ▶ Matlab HDL Coder
 - Independiente del fabricante de la FPGA
 - ▶ Herramientas específicas de cada fabricante
 - Xilinx System Generator for DSPs
 - DSP Builder for Intel® FPGAs (Altera DSP Builder)

- Puede utilizarse tanto desde Matlab como desde Simulink
 - ▶ Esta introducción está basada en Simulink
 - ▶ Debido a que principalmente se utiliza en forma mixta con la simulación de los convertidores
 - ▶ Permite integrar la lógica de control con la simulación del sistema
- Utilización de Punto Fijo
 - ▶ Punto Flotante se puede utilizar a expensas de una gran cantidad de recursos y latencia
- Discrete Solver
 - ▶ Fixed Step
- Configuraciones complejas de Matlab
 - ▶ HDL Workflow Advisor
 - Algebraic Loops, Target Architecture, etc.

Workflow Advisor

The screenshot shows the MATLAB/Simulink environment with a model named 'basic'. The 'Code' menu is open, and the 'HDL Workflow Advisor...' option is selected. The model diagram includes a constant block '1' (boolean) connected to 'in1' of an 'and' block, and a pulse generator block (double) connected to a 'boolean' block, which is then connected to 'in2' of the 'and' block. The 'and' block has two outputs: 'cnt' (ufix4) and 'boolean' (boolean), both connected to a scope block. The status bar at the bottom indicates 'Ready', '150%', and 'auto(VariableStepDiscrete)'.

basic - Simulink

File Edit View Display Diagram Simulation Analysis Code Tools Help

basic

basic

1 boolean

double boolean

in1 cnt

in2 and

ufix4 boolean

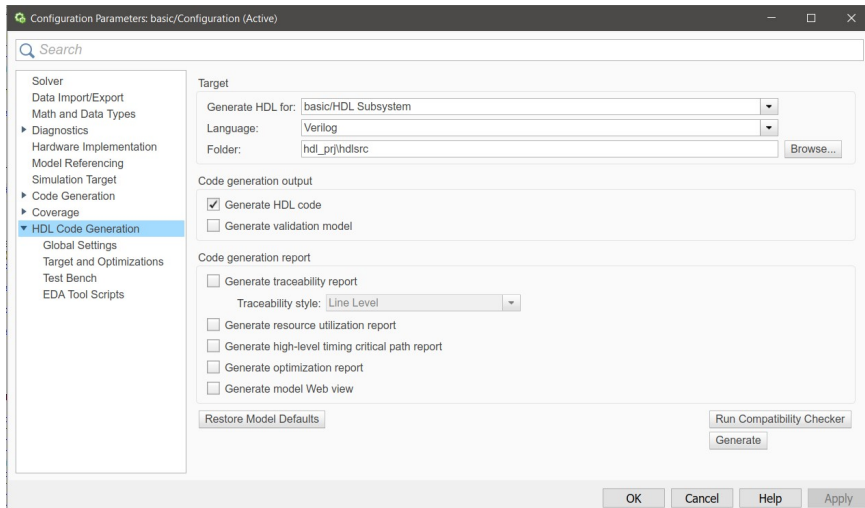
HDL Subsystem

Ready 150% auto(VariableStepDiscrete)

- C/C++ Code
- HDL Code
 - HDL Workflow Advisor...
 - Options...
 - Generate HDL
 - Generate Test Bench
 - Remove HDL Coder Configuration from Model
- PLC Code
- Data Objects
- External Mode Control Panel
- Simulink Code Inspector...
- Verification Wizards
- Polyspace
- DO Qualification Kit
- IEC Certification Kit

HDL Coder Properties (1)

- Botón Derecho en HDL Subsystem,
 - ▶ HDL Code → HDL Coder Properties ...



HDL Coder Properties (2)

Configuration Parameters: basic/Configuration (Active)

Search

- Solver
- Data Import/Export
- Math and Data Types
- ▶ Diagnostics
- ▶ Hardware Implementation
- ▶ Model Referencing
- ▶ Simulation Target
- ▶ Code Generation
- ▶ Coverage
- ▶ HDL Code Generation
 - Global Settings
 - Target and Optimizations
 - Test Bench
 - EDA Tool Scripts

Clock settings

Reset type: Asynchronous Reset asserted level: Active-high

Clock input port: clk Clock enable input port: clk_enable

Reset input port: reset Clock inputs: Single

Oversampling factor: 1 Clock edge: Rising

Additional settings

General Ports Coding style Coding standards Diagnostics Floating Point Target

Comment in header: <empty>

Verilog file extension: .v VHDL file extension: .vhd

Entity conflict postfix: _block Package postfix: _pkg

Reserved word postfix: _rsvd Split entity file postfix: _entity

Clocked process postfix: _process Split arch file postfix: _arch

Complex real part postfix: _re ☐ Split entity and architecture

Complex imaginary part postfix: _im VHDL architecture name: rtl

Enable prefix: enb Module name prefix: <empty>

Pipeline postfix: _pipe Timing controller postfix: _tc

VHDL library name: work

☐ Generate VHDL code for model references into a single library

Block generate label: _gen Output generate label: outputgen

Instance generate label: _gen Vector prefix: vector_of_

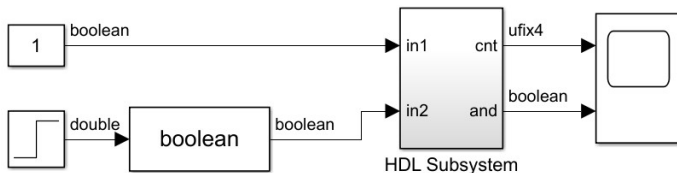
Instance prefix: u_ Instance postfix: <empty>

Prefix for the generated model name: gm_ Map file postfix: _map.txt

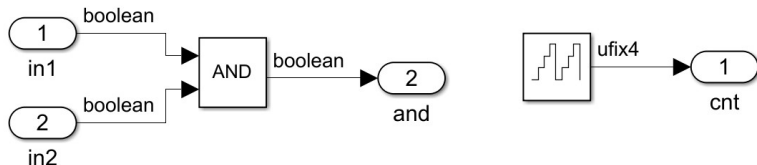
OK Cancel Help Apply

Ejemplo 1

- Modelo de simulación



- Detalle interno del bloque a sintetizar



Tipos de dato de Punto Fijo

- `ufixM_En_N`
 - ▶ Tipo de dato no signado de M bits en total con N bits de parte fraccionaria
 - ▶ Rango: $[0, 2^{M-N} - 2^{-N}]$
 - Ej. `ufix8_En2`: $[0, 2^6 - 2^{-2}] = [0, 63.75]$
- `sfix`
 - ▶ Tipo de dato signado de M bits en total con N bits de parte fraccionaria
 - ▶ Rango: $[-2^{M-N-1}, 2^{M-N-1} - 2^{-N}]$
 - Ej. `sfix8_En2`: $[-2^5, 2^5 - 2^{-2}] = [-32, 31.75]$
- Existen varios tipos predefinidos
 - ▶ Por ej. `uint8`, `int8`
- No es la única forma de definir un tipo de datos en punto fijo
 - ▶ Por ej. *Slope and Bias*

Ejemplo 1 - Código generado (1)

- Se muestran solo las partes relevantes del código
- Entradas y salidas denominadas de la misma forma que en Simulink
 - ▶ El diseño HDL que lo utilice solo debe instanciarlo
- Utiliza *enables*
 - ▶ Permite realizar fácilmente diversos *sample time*

- En el top.v

```
HDL_Subsystem HDL_Subsystem_inst(  
    .clk(clk),  
    .reset(rst_n),  
    .clk_enable(en),  
    .in1(j13),  
    .in2(j14),  
    .ce_out(),  
    .cnt(cnt),  
    .and_rsvd(j19));
```

```
module HDL_Subsystem  
    (clk ,  
     reset ,  
     clk_enable ,  
     in1 ,  
     in2 ,  
     ce_out ,  
     cnt ,  
     and_rsvd );  
  
    input  clk ;  
    input  reset ;  
    input  clk_enable ;  
    input  in1 ;  
    input  in2 ;  
    output ce_out ;  
    output [3:0] cnt ; // ufix4  
    output and_rsvd ;
```

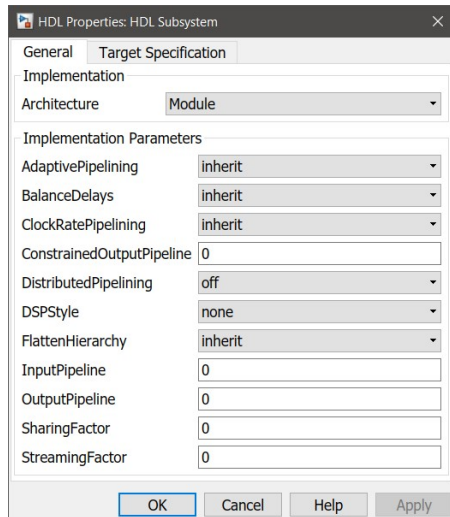
Ejemplo 1 - Código generado (2)

- La frecuencia de ce_out es la del modelo
 - ▶ Permite conectar en cascada diversos subsistemas

```
wire enb;
reg [3:0] Counter_Free_Running_out1; // ufix4
wire Logical_Operator_out1;
assign enb = clk_enable; // Free running, Unsigned Counter
// initial value = 0
// step value = 1
always @(posedge clk)
begin : Counter_Free_Running_process
    if (reset == 1'b0) begin
        Counter_Free_Running_out1 <= 4'b0000;
    end
    else begin
        if (enb) begin
            Counter_Free_Running_out1 <= Counter_Free_Running_out1 + 4'b0001;
        end
    end
end

assign cnt = Counter_Free_Running_out1;
assign Logical_Operator_out1 = in1 & in2;
assign and_rsvd = Logical_Operator_out1;
assign ce_out = clk_enable;
endmodule // HDL_Subsystem
```

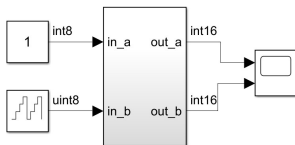
- Configuración del Reset
 - ▶ Sincrónicos vs Asíncrono
- Oversampling
- Optimization
 - ▶ Resource Sharing
 - ▶ Flatten Hierarchy
 - ▶ Sharing/Streaming Factor
 - ▶ Muchas mas
- Las optimizaciones pueden ser
 - ▶ Por Bloque
 - ▶ Por Diseño



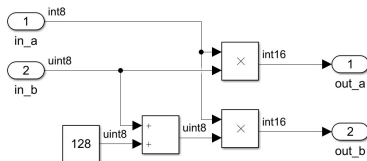
- Permite la simulación del código junto con el sistema a controlar
 - ▶ Simulación mixta
- Fácil verificación
 - ▶ Mediante la creación automática de TestBenchs
 - No incluido en el curso debido a restricciones de tiempo
- Ejemplo 2
 - ▶ Reutilización de hardware
 - Si el tiempo de muestreo del algoritmo es lento se puede multiplexar la utilización del hardware, MATLAB puede optimizarlo por sí solo
 - Existen ciertas restricciones
 - Por ej. que los bloques realicen exactamente la misma operación
 - Particularmente útil en operaciones de gran complejidad
 - Por ej. Multiplicación

Ejemplo 2

- Modelo de simulación



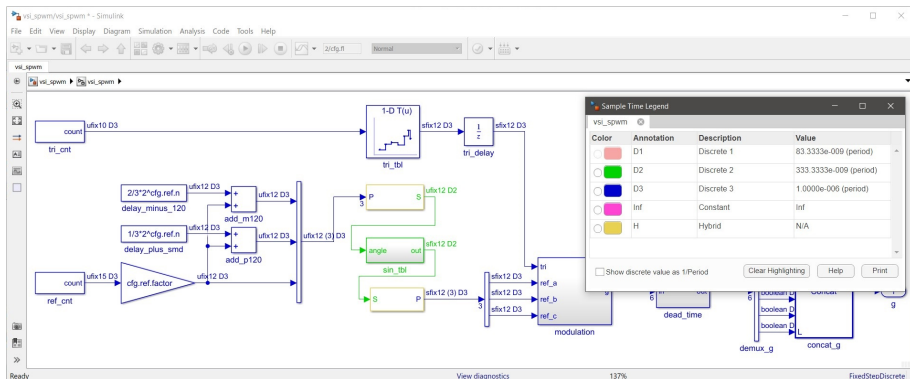
- Detalle interno del bloque a sintetizar



- Si $T_s = 1\mu s$
 - Al reutilizar el multiplicador el bloque subsistema requerirá un $T_s = 0.5\mu s$ ya que necesitará dos ciclos de clocks para realizar ambas multiplicaciones
- Ventajas
 - Se reduce la utilización de hardware
- Desventajas
 - Se necesitan mas ciclos
 - No siempre es una desventaja (por ej. en caso que existe *Oversampling* en otra parte del sistema)

Serialización y Deserialización

- Permiten la reutilización de hardware de forma simplificada
- Modifican el tiempo de muestreo de los bloques entre ellos
 - ▶ Debido al factor de relación entrada/salida
- Se observa en Display → Sample Time
- Utiliza T_s de la configuración de la simulación
 - ▶ No hay necesidad de generar los en en código HDL



- Rate Transition
 - ▶ Simulink → Signal Attribute
- Repeat
 - ▶ DSP System Toolbox HDL Support
- Zero Order Hold con diferente tiempo de muestreo
- Forzar el tiempo de muestreo de cualquier bloque posterior
 - ▶ etc ...
- Mas información en [HDL Coder](#)
 - ▶ [User guide](#)
 - ▶ [Getting started](#)
 - ▶ [Reference](#)

Integración (2)

- Operación utilizada en la mayoría de los sistemas de control
- Forward Euler

$$H(z) = T_s \frac{z^{-1}}{1 - z^{-1}} \Rightarrow y_n = y_{n-1} + T_s u_{n-1}$$

- Backward Euler

$$H(z) = T_s \frac{1}{1 - z^{-1}} \Rightarrow y_n = y_{n-1} + T_s u_n$$

- Trapezoidal

- ▶ Promedio entre Backward Euler y Forward Euler

$$H(z) = T_s \frac{1 + z^{-1}}{1 - z^{-1}} \Rightarrow y_n = y_{n-1} + \frac{T_s}{2} (u_n + u_{n-1})$$

- Lazos algebraicos ...

- ▶ Forward Euler es simple de implementar ya que solo depende de los estados en el instante de tiempo $n - 1$