ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

# Real-Time Bird Audio Detection using AI on FPGAs

Mestrado em Engenharia Informática e de Computadores


Rodrigo Lopes da Silva

Orientadores:

Professor Doutor Rui Duarte

Professor Doutor Mário Véstias

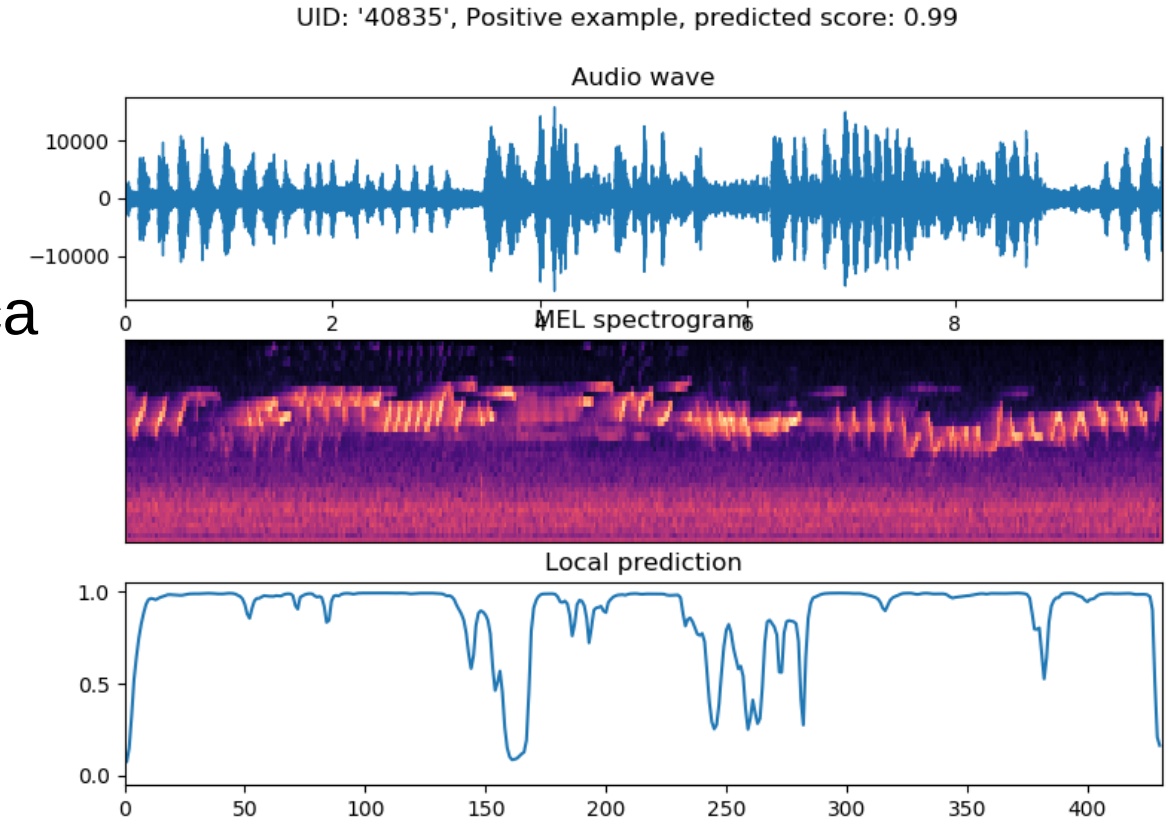

2024-07-27

# Introdução

- Motivações
  - Aves ajudam a monitorizar biodiversidade e padrões migratórios
  - Desafios no levantamento de dados visuais
  - Áudio permite recolha de dados sem perturbações
  - Redes Neurais em CPU e/ou GPU são exigentes em energia e espaço
  - Mobilidade de um sistema embebido com FPGA
- Objectivo
  - Adaptar um modelo do “Bird Audio Detection Challenge” para hardware
  - Estudar a quantização de modelos TensorFlow
  - Benchmark da implementação em hardware versus software

# Bird Audio Detection Challenge (BAD)

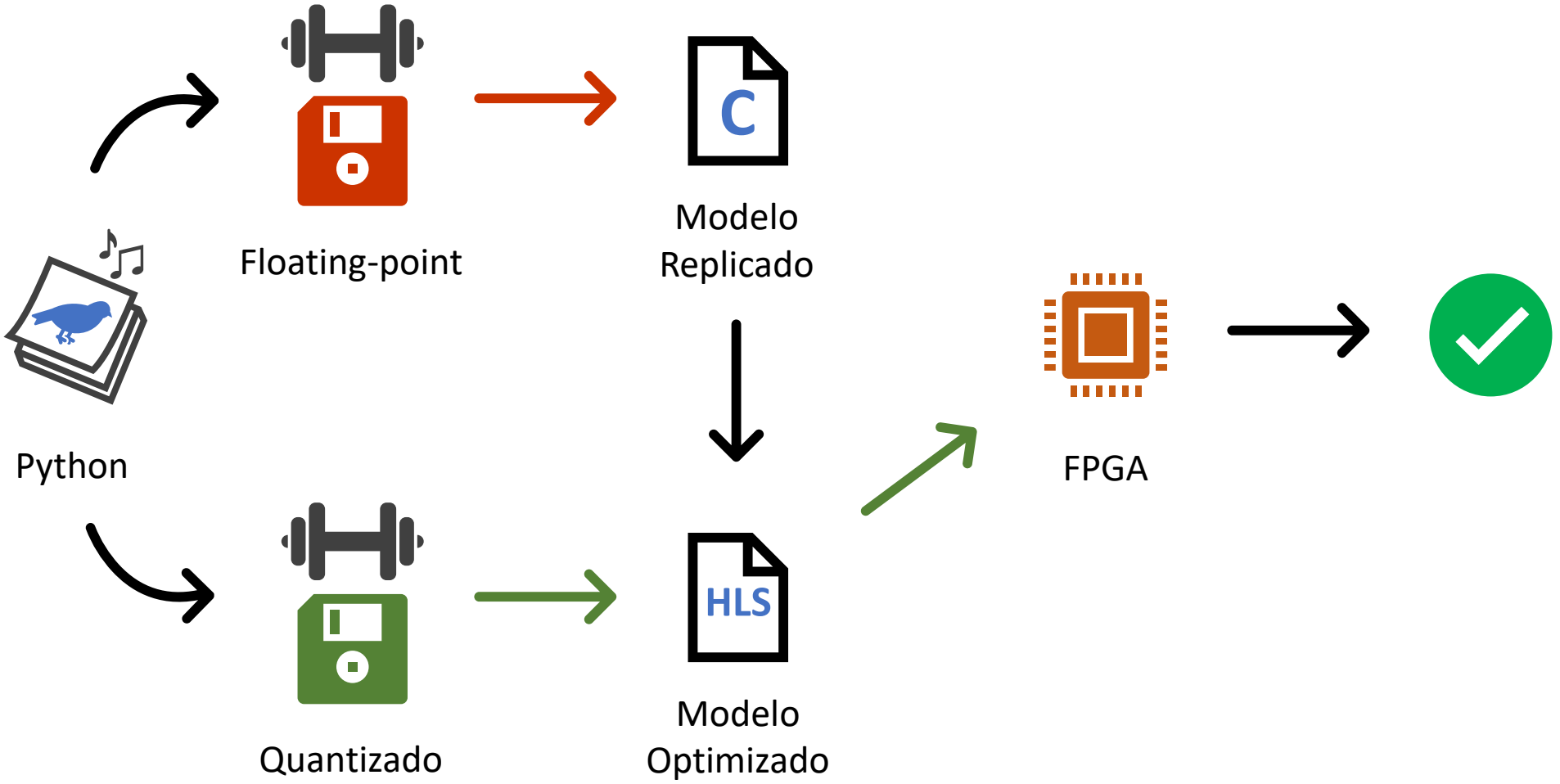
- Queen Mary University of London
- 2016 a 2018
- Concurso com submissões de algoritmos que detetam a presença de sons emitidos por pássaros
- Datasets:
  - Ficheiros WAV de 10 segundos
  - 3 para treino
  - 3 para avaliação



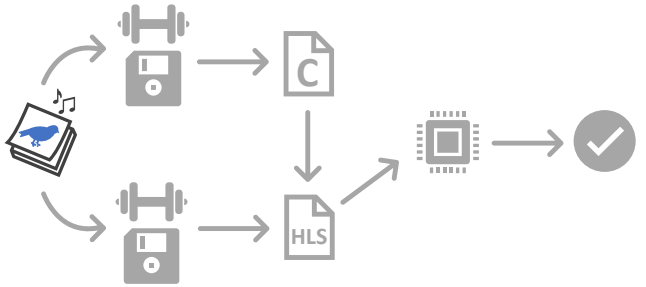
# Bird Audio Detection Challenge (BAD)

| Nome | Gravações | Tipo de sons | Localização | Utilização |
|---|---|---|---|---|
| BirdVox-DCASE-20k | 20000 | Vida selvagem, Pássaros (50%), Noite | Ithaca (NY), USA | Avaliação |
| Chernobyl | 6620 | Vida selvagem, Pássaros, Meteorológicos | Chernobyl EZ | Treino |
| freefield1010 | 7690 | Cidade, Vida selvagem, Pássaros | Várias | Treino |
| PolandNFC | 4000 | Veados, Mar, Pássaros, Meteorológicos | Polónia | Avaliação |
| Warblrb10k | 8000 | Cidade, Pessoas (com imitações), Pássaros | Inglaterra | Treino |
| Warblrb10k | 2000 | Cidade, Pessoas (com imitações), Pássaros | Inglaterra | Avaliação |

# Fluxo do Projecto

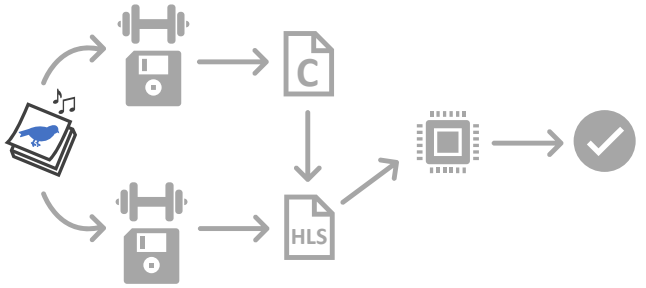


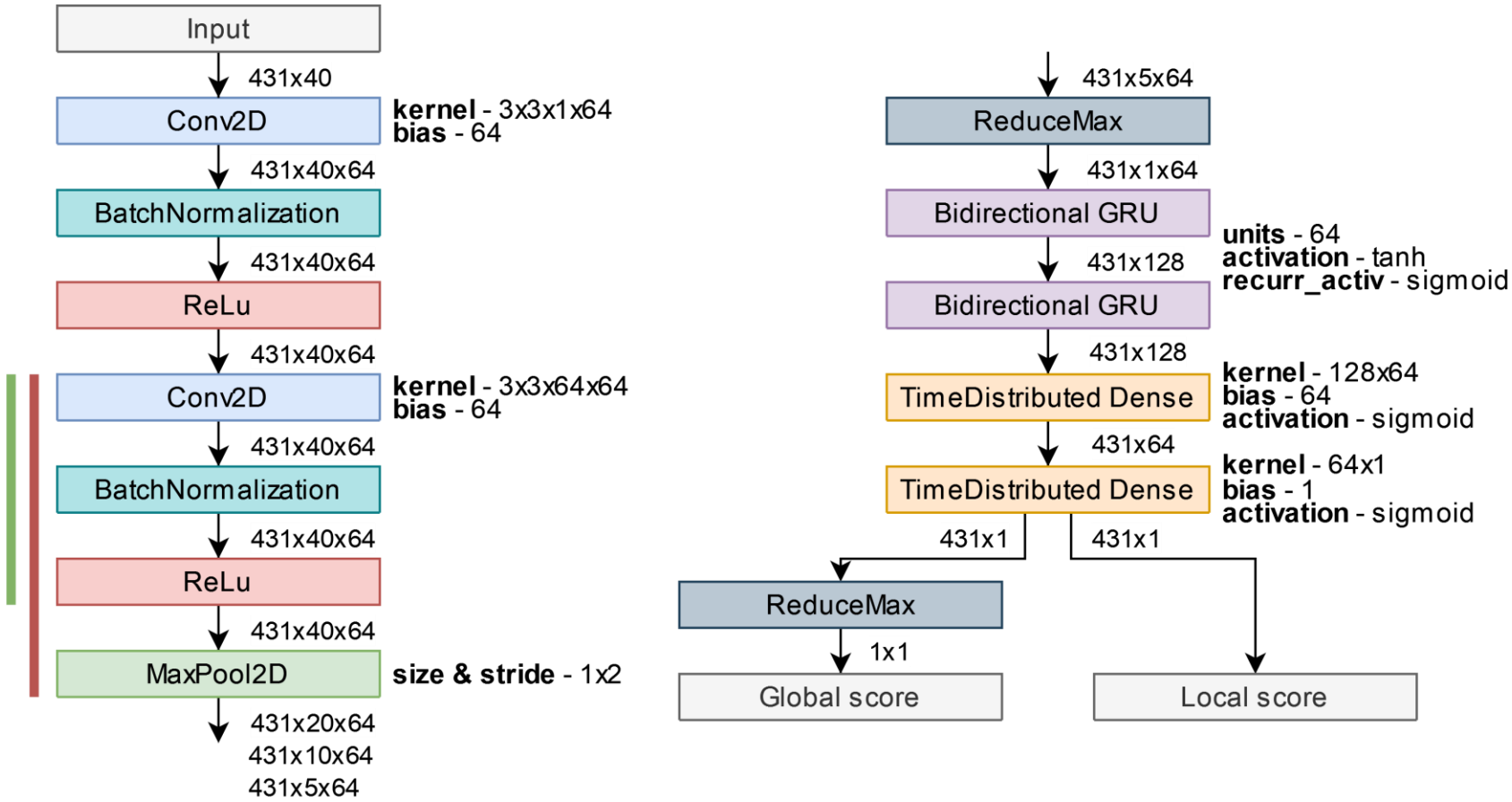
# Selecção de algoritmos



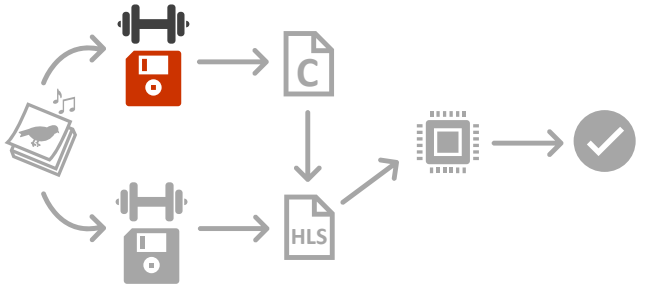

- Problemas na escolha de algoritmos:
  - Falta de documentação para por a funcionar os algoritmos encontrados
  - Versões antigas do Python
  - Versões antigas das bibliotecas usadas
- Soluções para as dificuldades encontradas:
  - Actualização para versões mais recentes do Python e bibliotecas
  - Sucesso para 2 algoritmos encontrados:
    - All-Conv-Net-for-BAD
    - microfaune-ai

# microfaune-ai

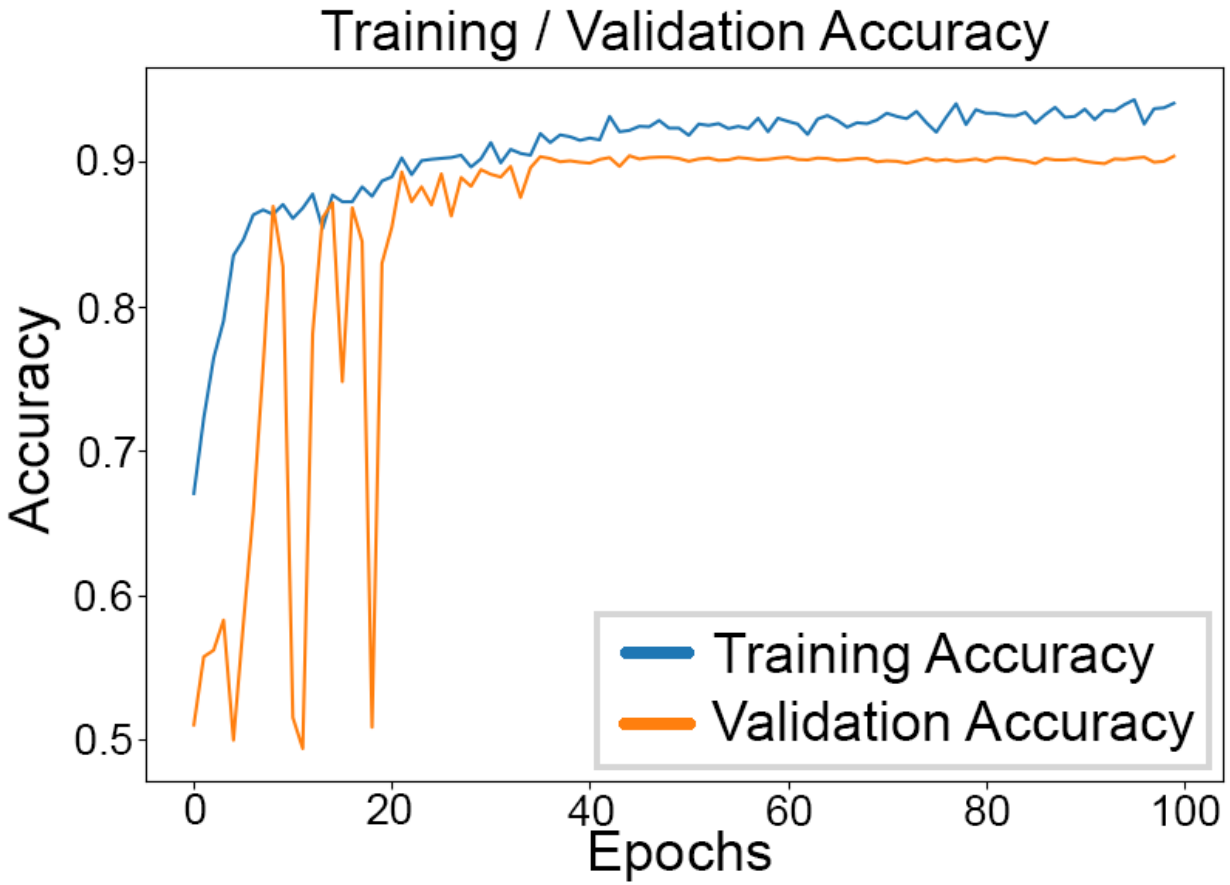


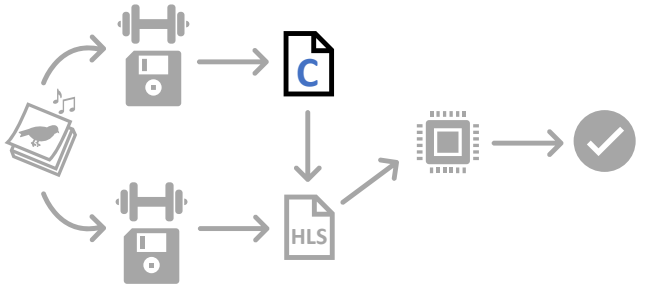
# Treino e Pesos - Floating-point




- Porquê treinar?
  - Se é possível treinar, então Treino com Quantização é uma possibilidade
  - Validar a precisão do modelo segundo a sua documentação
  - Extrair os pesos
  - Validar a implementação em C
- Precisão reportada na documentação:
  - 90.2%
- Precisão pós treino:
  - 90.5%

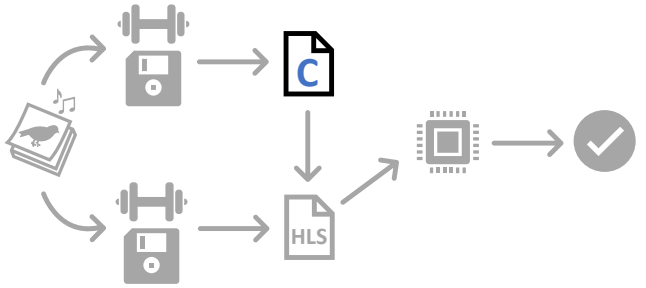


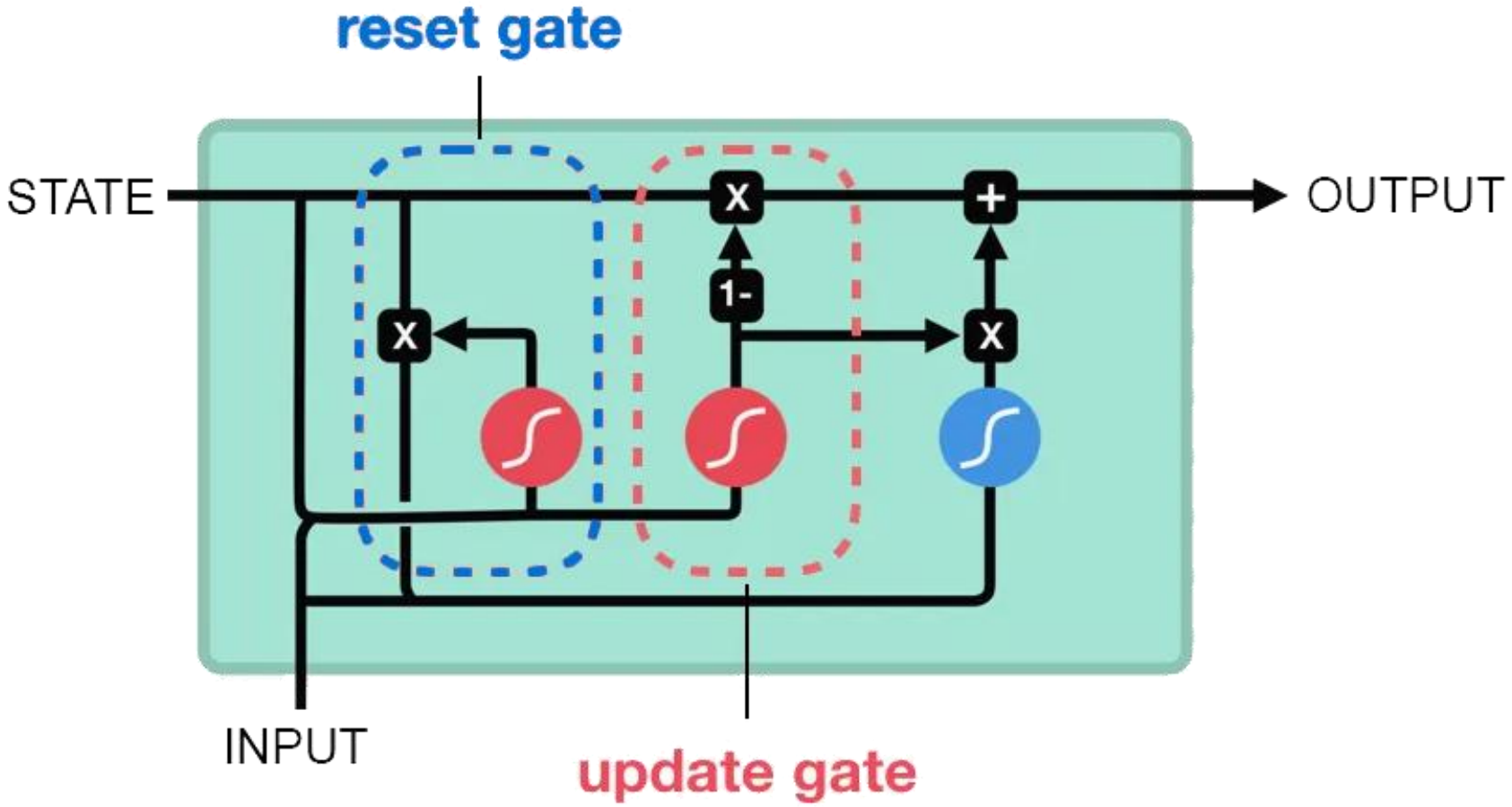
# Replicação do modelo em C



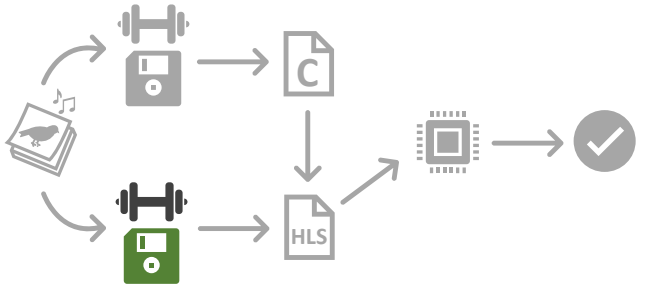

| Python | Python | C |
| --- | --- | --- |
| TensorFlow | Numpy | Replicação do modelo |
| Debug complicado | Debug fácil e simples | Uso de pesos Floating-point |
| *backend.print_tensor* | Controlo da ordem dos prints | |
| Prints fora de ordem | | |

# Replicação da célula GRU

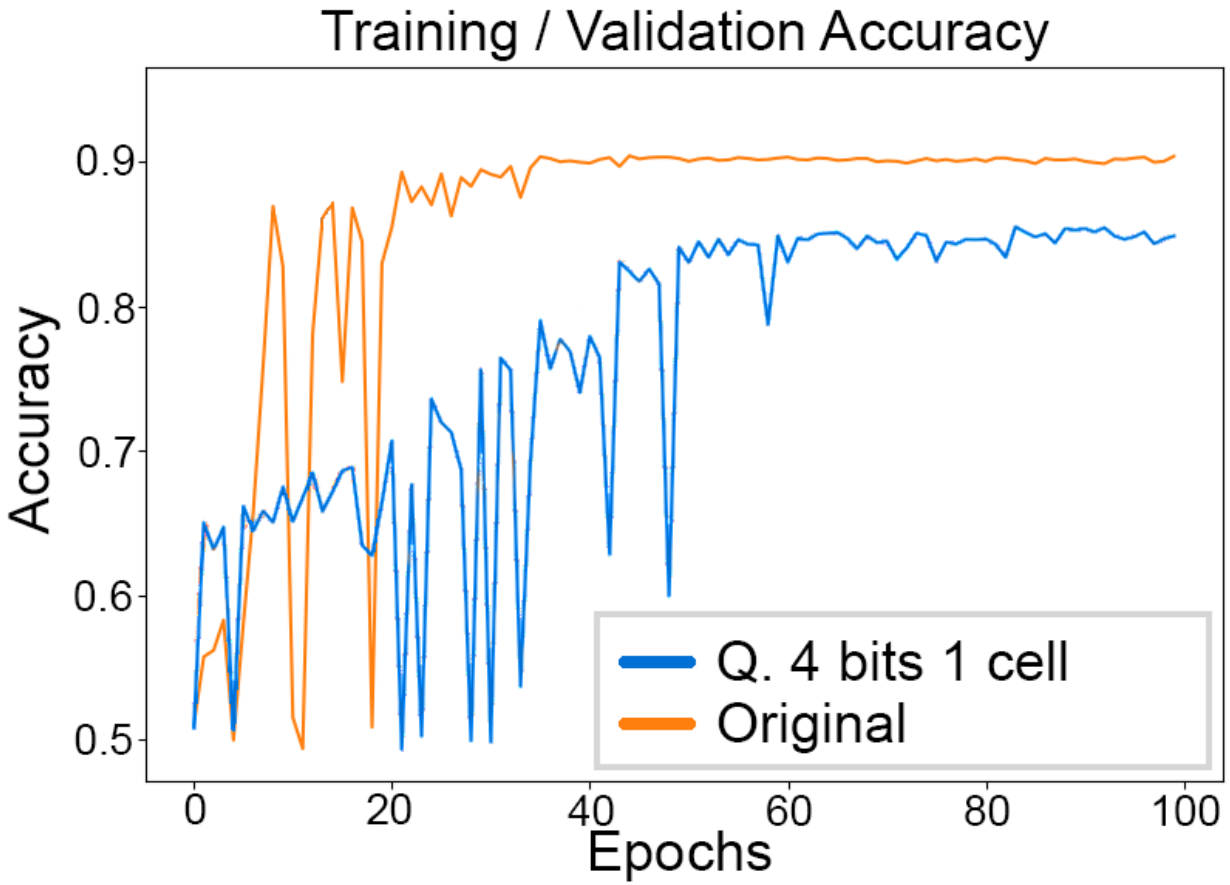


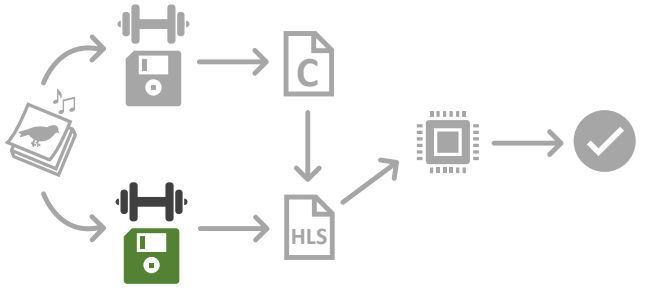
# Treino e Pesos - Quantização



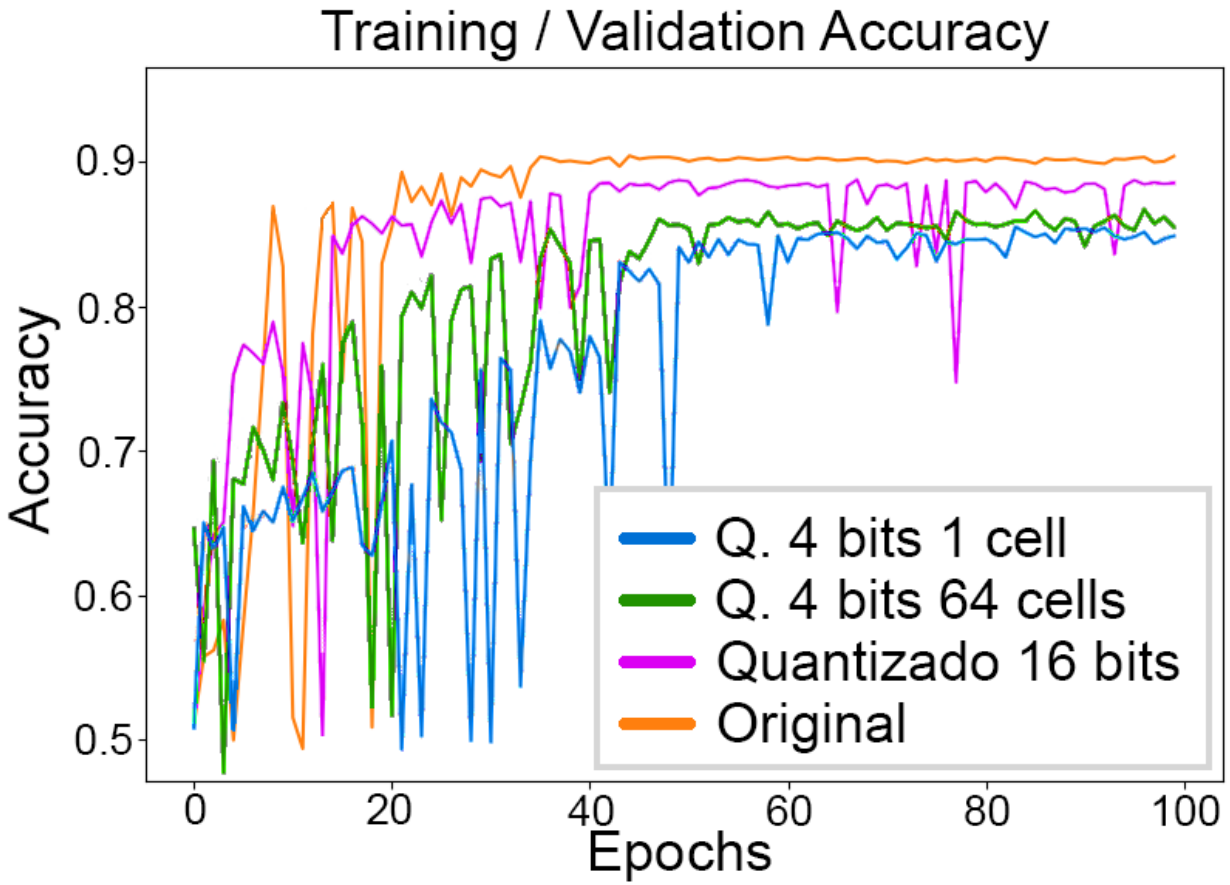

- Treino com Quantização
- QKeras
  - Biblioteca extensão ao Keras
  - Substituição de layers “Q----”
  - Quantizações independentes entre layers, pesos, activações
- microfaune_ai
  - Adicionado ao ínicio do modelo:
    - QActivation com ReLU → Q(4,0)
  - Conv2D e BatchNormalization
    - QConv2DBatchnorm → Q(4,1)
  - Activação ReLU
    - QActivation com ReLU → Q(4,0)
  - Bidirectional GRU
    - Quantização por truncagem → Q(8,1)
  - Time-Distributed Dense, floating-point



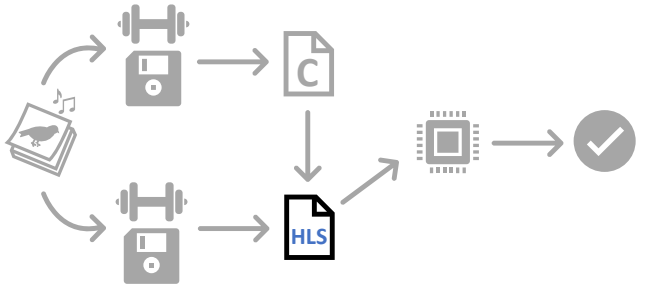
# Treino e Pesos - Quantização




| Modelo | Validation Accuracy |
|---|---|
| Original | 90.50 % |
| Quantizado 16 bits | 88.60 % |
| Quantizado 8 bits | 85.97 % |
| Quantizado 4 bits | 85.46 % |
| Quantizado 4 bits / 1 cell | 84.91 % |
| Quantizado 2 bits | 65.07 % |

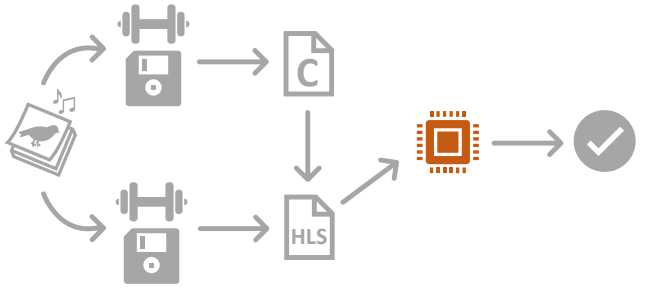


# High-Level Synthesis

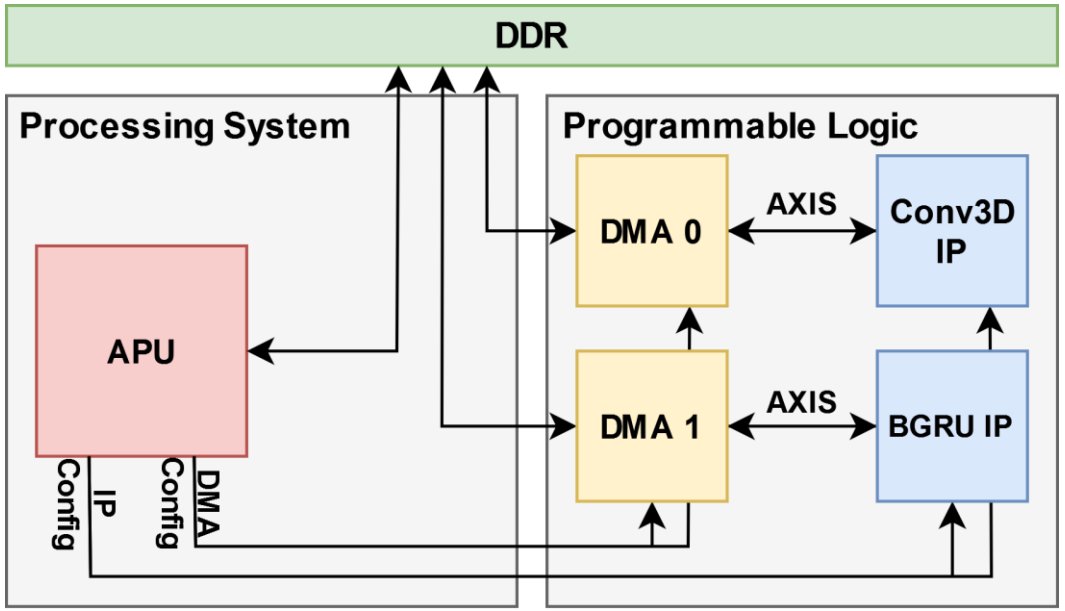



- Ajustes ao código C para migrar para HLS
  - Substituição das transferências de dados entre layers:
    - Pointeiros → AXI Lite e AXI Stream
  - Substituição de Pesos:
    - Floating-point → Quantizados (CNN 4 bits, RNN 8 bits, Pós-RNN Floating-point)
  - Adicionado pós-processamento entre secções e algumas layers do modelo:
    - CNN 4bits → RNN 8bits
    - Outputs Forward GRU e Backward GRU transformados em Bidirectional GRU
    - RNN 8bits → Pós-RNN Floating-point
- Optimizações exploradas:
  - Alterada 1ª Convolução de 1 Filtro e 64 Channels, para 64 Filtros e 64 Channels: 3x3x1x64 → 3x3x64x64
  - 64 acumulações em paralelo nas Convoluções

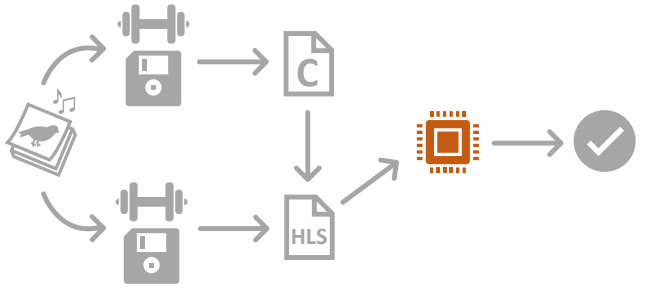
# Arquitectura proposta Hardware / Software

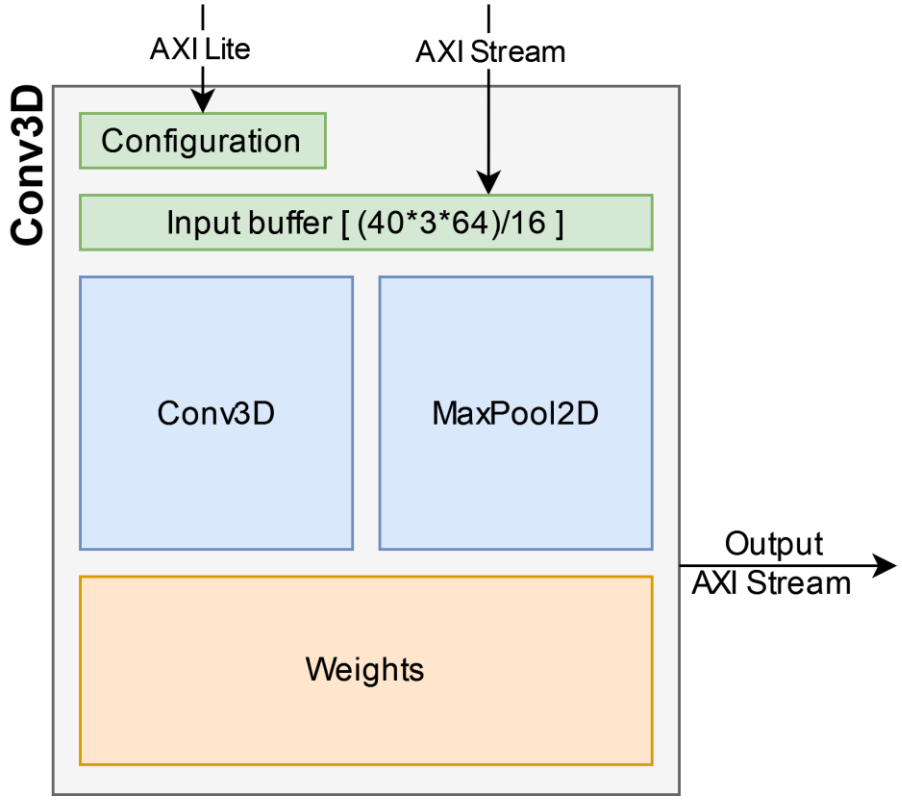



- Processing System
  - Lógica de utilização do modelo
  - Configura os blocos hardware
  - Executa layers não quantizadas
  - Coloca input e recolhe output das layers hardware da memória externa
  - Pós-processamento, conversão de dados entre secções do modelo
- Programmable Logic
  - Executa layers quantizadas
  - 2 blocos hardware, Conv3D e BGRU
  - DMAs recolhem input e colocam output das layers hardware na memória externa

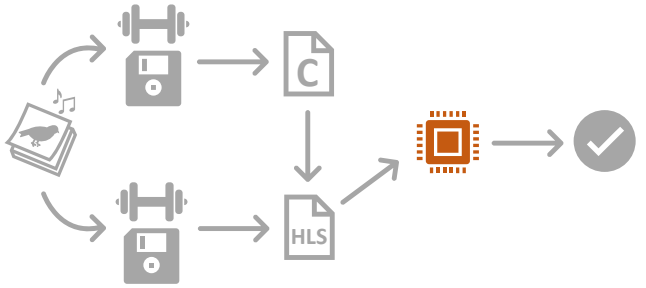


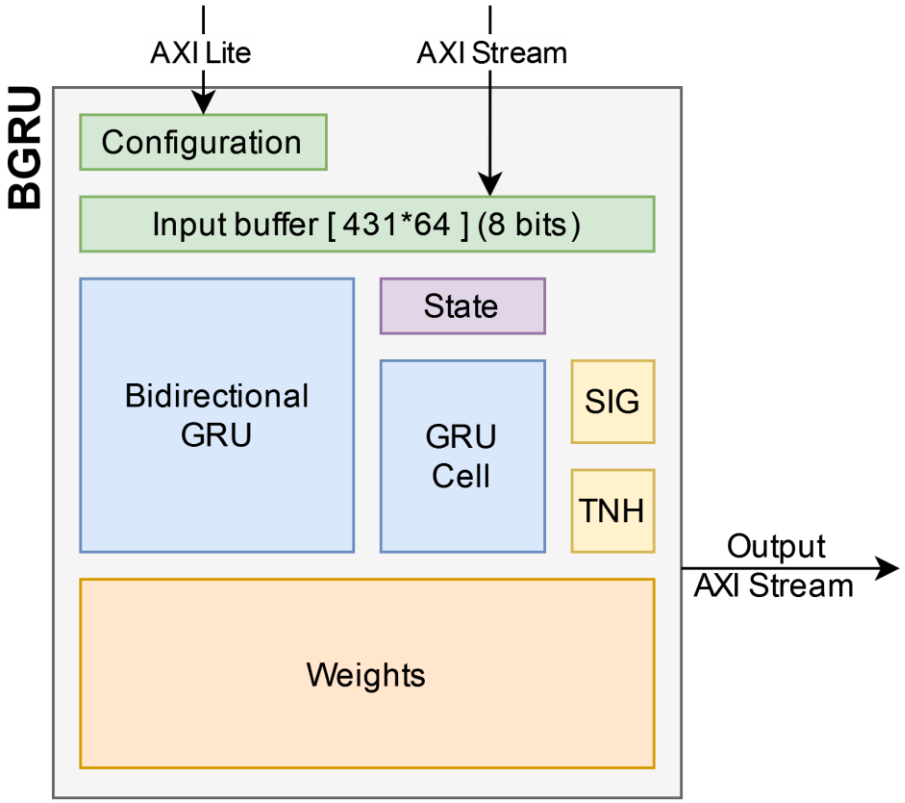
# Blocos IP e Auxiliares



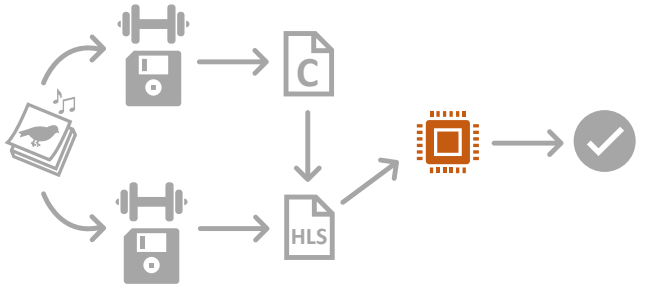



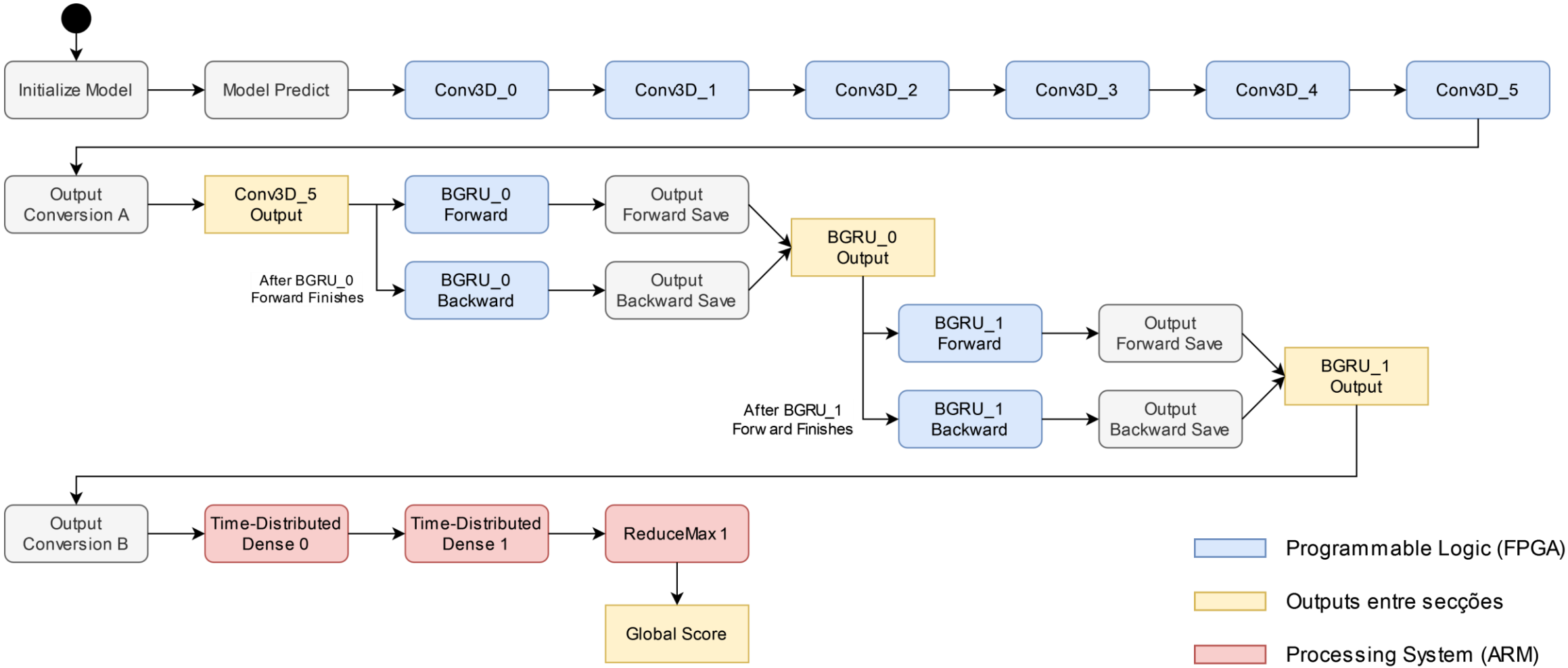
# Blocos IP e Auxiliares



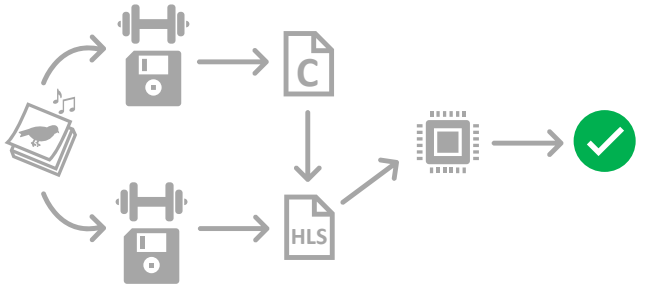



# Design Software



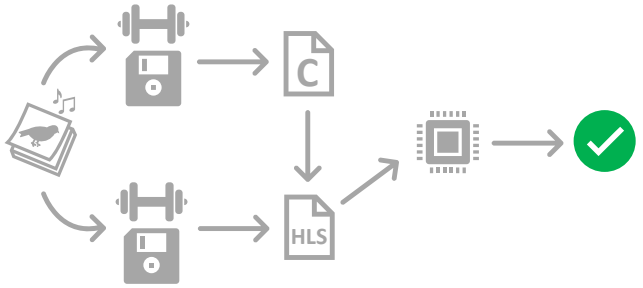
# Resultados




- Modelos comparados
  - **Original**: Python, floating-point, 64 cells
  - **Modified**: Python, floating-point, 1 cell
  - **QKeras**: Python, CNN 4 bits, 1 cell
  - **Optimized**: QKeras em HLS, RNN 8 bits por truncagem

| Modelo | Precisão (400, 50/50) |
|---|---|
| Original | 89.75% |
| Modified | 86.25% |
| QKeras | 80.00% |
| **Optimized** | **79.50%** |

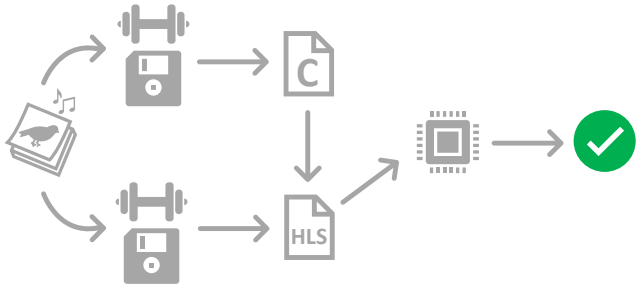
# Desempenho




| *ms* | CNN | RNN | Software | Total |
|---|---|---|---|---|
| ARM | 53477 | 10.52 | 7.4 | 53495 |
| **Optimized** | **666** | **4.65** | **7.4** | **679** |
| Speed Up | 8025% | 226% | 100% | 7877% |

Comparação de desempenho Software vs Optimizado

# Recursos



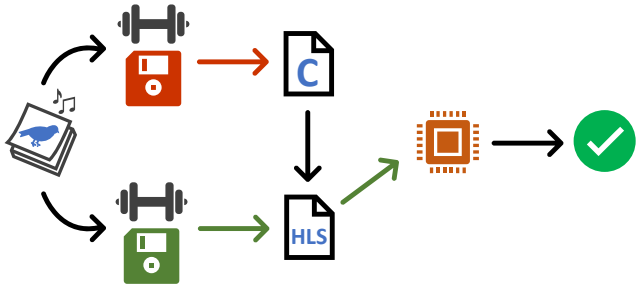

| *FPGA* | BRAM | DSP | FF | LUT |
|---|---|---|---|---|
| **Optimized** | **101** | **0** | **13971** | **14366** |
| ZU3CG | 216 | 360 | 141120 | 70560 |
| **Usage %** | **46%** | **0%** | **9.9%** | **20.36%** |

Recursos da FPGA usados

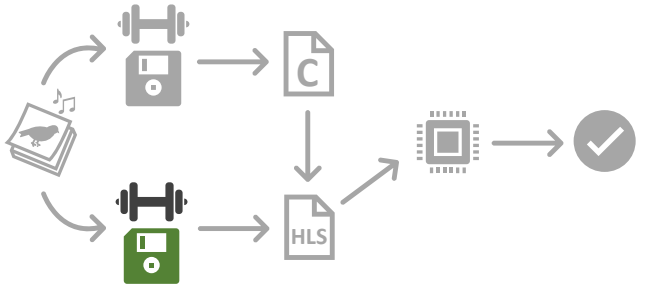
# DEMO

# Slides Auxiliares

# SLIDE – MAPA colorido



# Layers QKeras




```python
import numpy as np
from tensorflow import keras

spec = keras.Input(
    shape=[None, 40, 1], dtype=np.float32
)


x = keras.Conv2D(
    64, (3, 3), padding="same",
    activation=None
)(spec)
x = keras.BatchNormalization(momentum=0.95)(x)

x = keras.ReLU()(x)
```

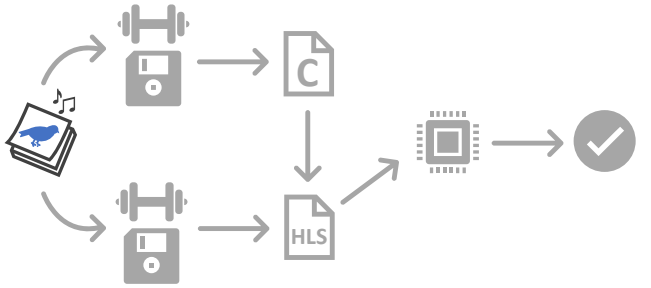
```python
import numpy as np
from qkeras import QConv2DBatchnorm, QActivation

spec = keras.Input(
    shape=[431, 40, 1], dtype=np.float32
)
x = QActivation(f"quantized_relu(4,0)")(spec)
#
x = QConv2DBatchnorm(
    64, (3, 3), padding="same", activation=None,
    kernel_quantizer = f"quantized_bits(4,1,1)",
    bias_quantizer = f"quantized_bits(4,1,1)",
    momentum=0.95
)(x)
x = QActivation(f"quantized_relu(4,0)")(x)
```

# GRU - Tensorflow

```python
kernel = []
recurrent_kernel = []
bias = []

input_bias, recurrent_bias = tf.unstack(bias)

def step(cell_inputs, cell_states):
    """Step function that will be used by Keras RNN backend."""
    h_tm1 = cell_states[0]

    # inputs projected by all gate matrices at once
    matrix_x = backend.dot(cell_inputs, kernel)
    matrix_x = backend.bias_add(matrix_x, input_bias)

    x_z, x_r, x_h = tf.split(matrix_x, 3, axis=1)

    # hidden state projected by all gate matrices at once
    matrix_inner = backend.dot(h_tm1, recurrent_kernel)
    matrix_inner = backend.bias_add(matrix_inner, recurrent_bias)

    recurrent_z, recurrent_r, recurrent_h = tf.split(matrix_inner, 3, axis=1)

    z = tf.sigmoid(x_z + recurrent_z)
    r = tf.sigmoid(x_r + recurrent_r)
    hh = tf.tanh(x_h + r * recurrent_h)

    # previous and candidate state mixed by update gate
    h = z * h_tm1 + (1 - z) * hh
    return h, [h]
```

```cpp
void gru_tensorflow(gruval* input, gruval* state, gruval* output) {
    // state = (h_tm1 = cell_states[0])

    /* inputs projected by all gate matrices at once */
    // matrix_x = backend.dot(cell_inputs, kernel)
    for (int i = 0; i < KERNEL_COLS; ++i) {
        matrix_x[i] = 0;
        for (int j = 0; j < KERNEL_ROWS; ++j) {
            gruval iVal = input[j];
            gruval kVal = kernel[j][i];
            matrix_x[i] += iVal * kVal;
        }
    }
    // matrix_x = backend.bias_add(matrix_x, input_bias)
    for (int i = 0; i < KERNEL_COLS; ++i) { matrix_x[i] += bias[i]; }

    // x_z, x_r, x_h = tf.split(matrix_x, 3, axis=1)
    gruval* x_z = matrix_x + (SPLIT_SIZE * 0);
    gruval* x_r = matrix_x + (SPLIT_SIZE * 1);
    gruval* x_h = matrix_x + (SPLIT_SIZE * 2);

    /* hidden state projected by all gate matrices at once */
    // matrix_inner = backend.dot(h_tm1, recurrent_kernel)
    for (int i = 0; i < KERNEL_COLS; ++i) {
        matrix_inner[i] = 0;
        for (int j = 0; j < KERNEL_ROWS; ++j) {
            gruval iVal = state[j];
            gruval kVal = recurrent_kernel[j][i];
            matrix_inner[i] += iVal * kVal;
        }
    }
    // matrix_x = backend.bias_add(matrix_inner, recurrent_bias)
    for (int i = 0; i < KERNEL_COLS; ++i) { matrix_inner[i] += recurrent_bias[i]; }

    // recurrent_z, recurrent_r, recurrent_h = tf.split(matrix_inner, 3, axis=1
    gruval* recurrent_z = matrix_inner + (SPLIT_SIZE * 0);
    gruval* recurrent_r = matrix_inner + (SPLIT_SIZE * 1);
    gruval* recurrent_h = matrix_inner + (SPLIT_SIZE * 2);

    // z = tf.sigmoid(x_z + recurrent_z)
    for (int i = 0; i < SPLIT_SIZE; ++i) { z[i] = sigmoid(x_z[i] + recurrent_z[i]); }
    // r = tf.sigmoid(x_r + recurrent_r)
    for (int i = 0; i < SPLIT_SIZE; ++i) { r[i] = sigmoid(x_r[i] + recurrent_r[i]); }
    // hh = tf.tanh(x_h + r * recurrent_h)
    for (int i = 0; i < SPLIT_SIZE; ++i) { hh[i] = tanh(x_h[i] + (r[i] * recurrent_h[i])); }

    // previous and candidate state mixed by update gate
    for (int i = 0; i < OUTPUT_SIZE; ++i) { output[i] = z[i] * state[i] + (1 - z[i]) * hh[i]; }
}
```

# Selecção de algoritmos

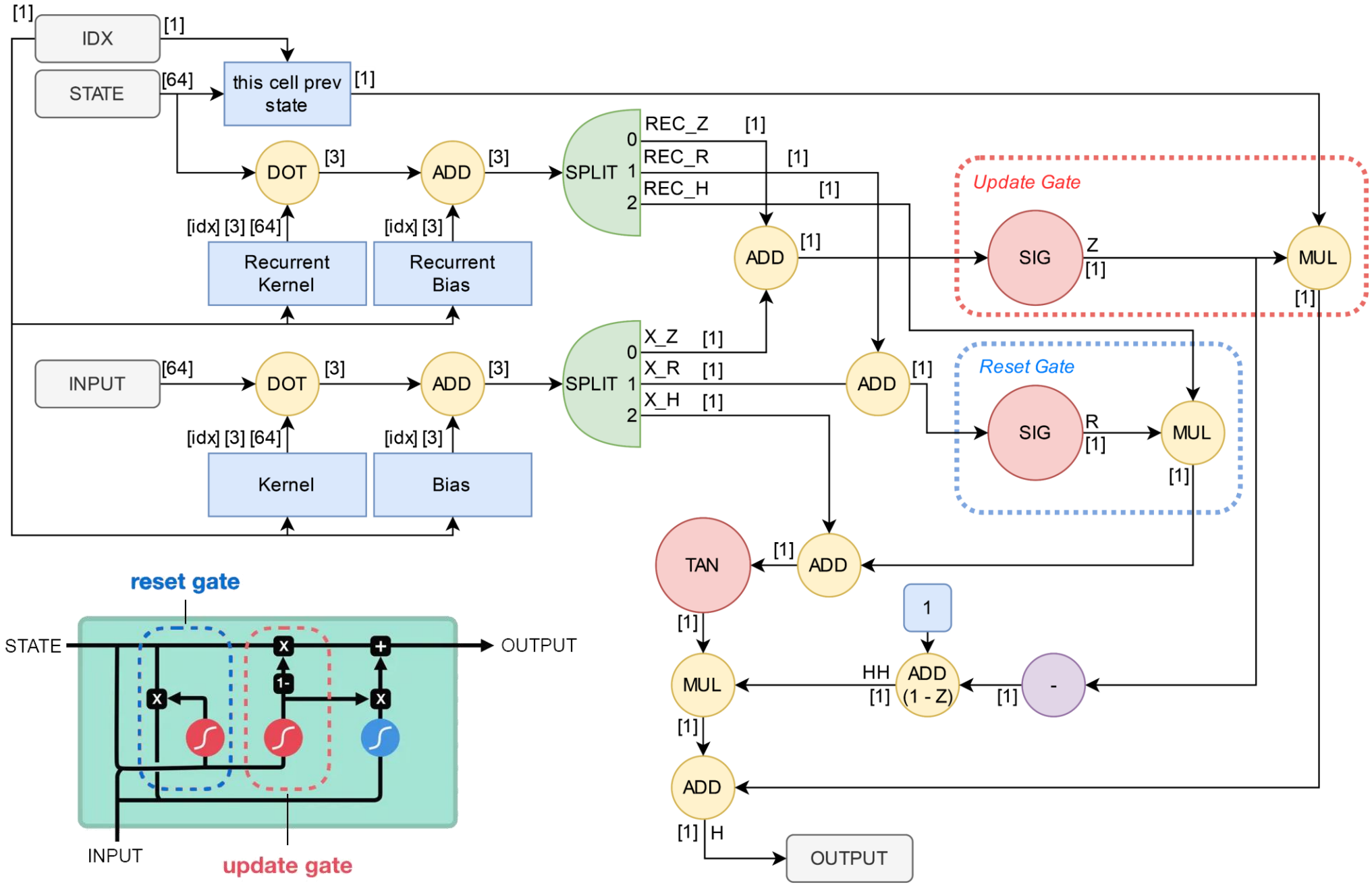


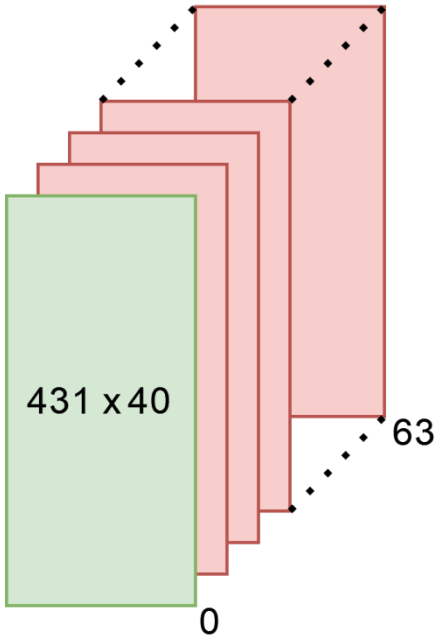

**All-Conv-Net-for-BAD**

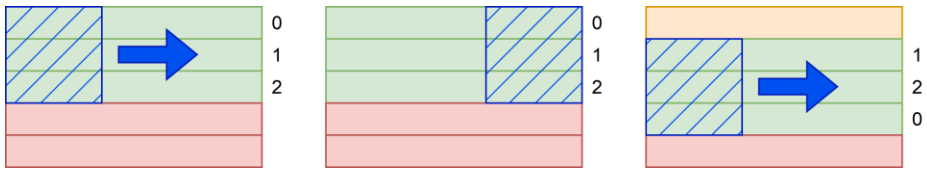
- CNN
- Layers:
  - Conv2D
  - BatchNormalization
  - MaxPooling2D, (*LPV)*
  - Dense, *(MPV)*
  - Conv1D, *(LPV)*
- Variantes e taxas de precisão:
  - Max Pool Variant (MPV)
    - 85.0 % - *Paper*
    - 87.9 % - Treinado
  - Learned Pool Variant (LPV)
    - 88.9 % - *Paper*
    - 86.1 % - Treinado

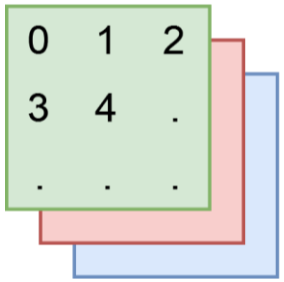
**microfaune-ai**

- CNN e RNN
- Layers:
  - Conv2D
  - BatchNormalization
  - MaxPooling2D
  - ReduceMax
  - Bidirectional + GRU
  - TimeDistributed + Dense
- Taxas de precisão:
  - 90.2 % - Reportado na documentação
  - 90.5 % - Treinado

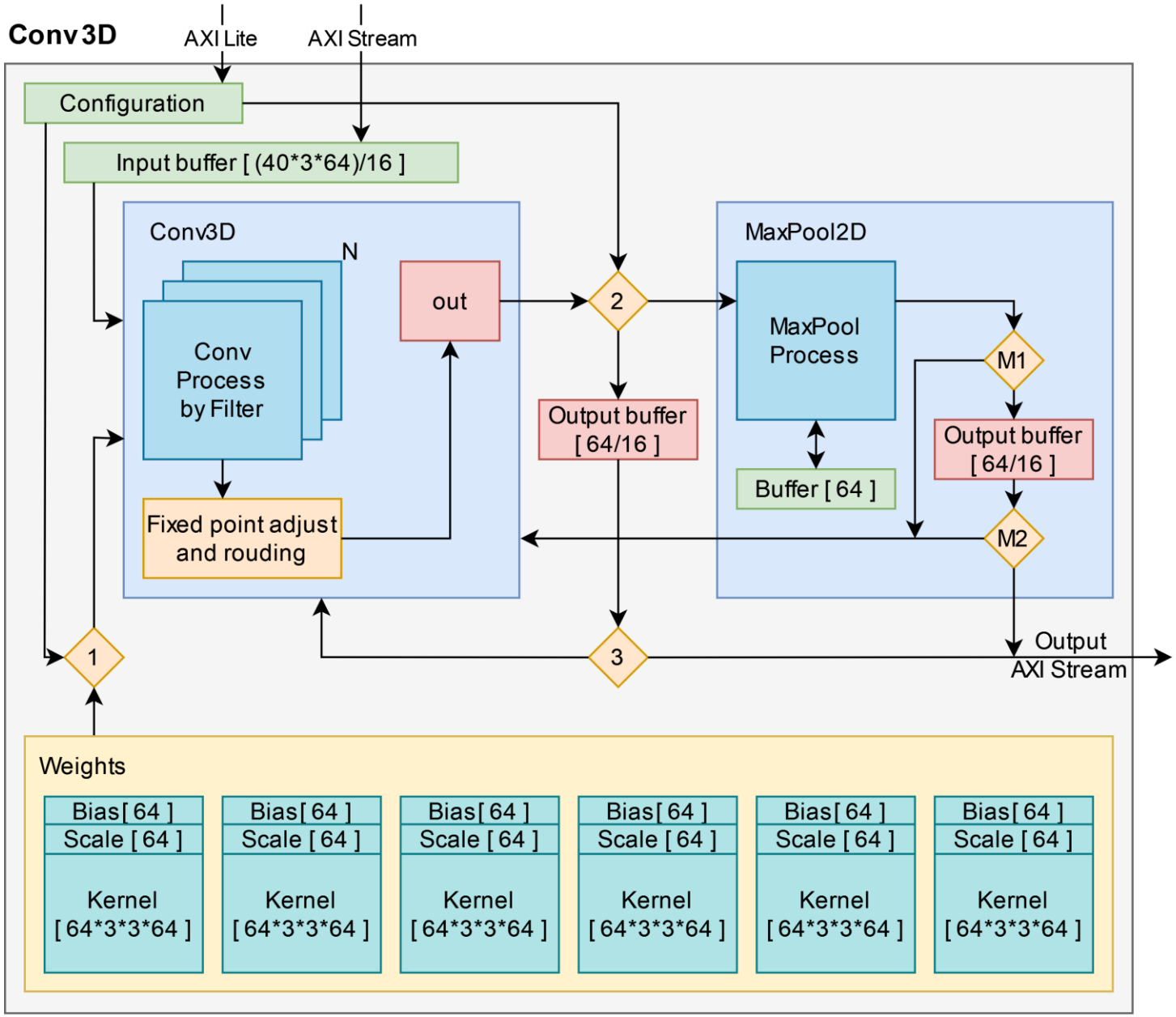

IDX
STATE
this cell prev state
DOT
ADD
SPLIT
REC_Z
REC_R
REC_H
Recurrent Kernel
Recurrent Bias
INPUT
Kernel
Bias
X_Z
X_R
X_H
Update Gate
Reset Gate
SIG
MUL
ADD
TAN
1
HH
ADD (1 - Z)
-
H
OUTPUT
reset gate
update gate
STATE
INPUT
OUTPUT


431 x 40
63
0


0
1
2
0
1
2
1
2
0


0 1 2
3 4 .
. . .


0 0 0 1 1 1 2 ...

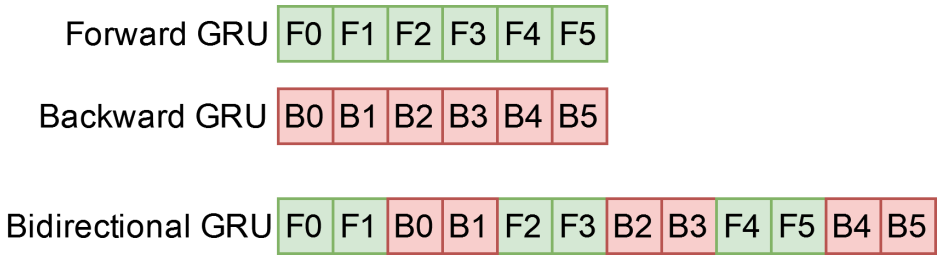

Conv3D
AXI Lite
AXI Stream
Configuration
Input buffer [ (40*3*64)/16 ]
Conv3D
N
Conv Process by Filter
out
Fixed point adjust and rouding
2
Output buffer [ 64/16 ]
MaxPool2D
MaxPool Process
Buffer [ 64 ]
M1
Output buffer [ 64/16 ]
M2
1
3
Output AXI Stream
Weights
Bias[ 64 ]
Scale [ 64 ]
Kernel [64*3*3*64 ]

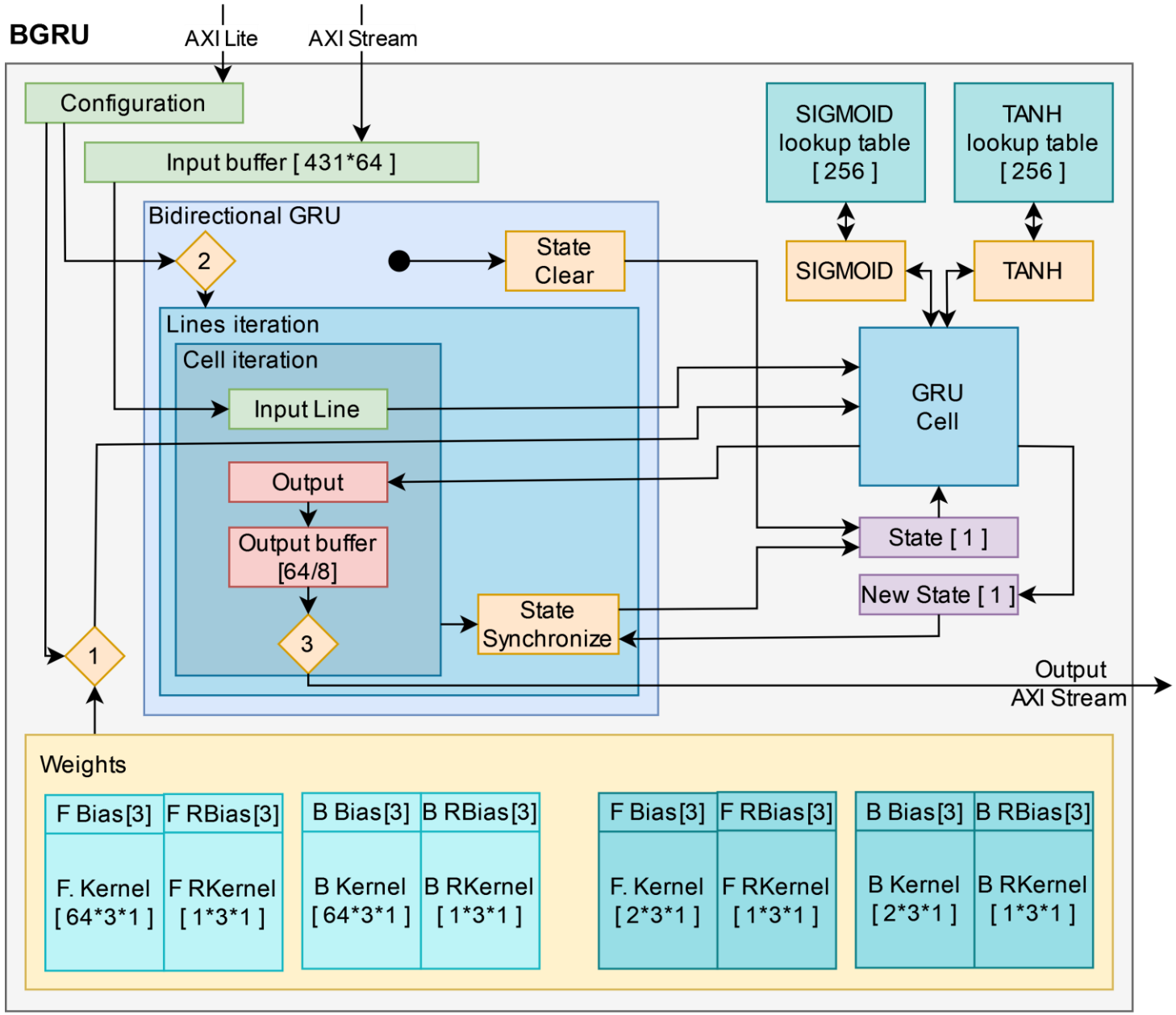
| *Conv3D Unroll* | K. lines | K. cols | 16 acc | Cycles | BRAM | DSP | FF | LUT |
|---|---|---|---|---|---|---|---|---|
| **Ultra96-V2** | - | - | - | - | 216 | 360 | 141120 | 70560 |
| **Baseline** | 1 | 1 | 1 | 52961606 | 79 | 0 | 881 | 7161 |
| | 1 | 1 | 2 | 33101126 | 79 | 0 | 885 | 8017 |
| **Selected** | 1 | 1 | 4 | 22067526 | 150 | 0 | 1138 | 9392 |
| | 1 | 3 | 1 | **101509446** | 79 | 0 | 1087 | 9337 |
| | 1 | 3 | 2 | **84959046** | 79 | 0 | 1088 | 12179 |
| | 1 | 3 | 4 | 15447366 | **458** | 0 | 1618 | 16256 |
| | 3 | 1 | 1 | **62891847** | 79 | 0 | 1104 | 9700 |
| | 3 | 1 | 2 | 43031367 | 79 | 0 | 1062 | 12604 |
| | 3 | 1 | 4 | 33101126 | 150 | 0 | 1803 | 17519 |
| | 3 | 3 | 1 | **95992646** | 79 | 0 | 1519 | 16561 |
| | 3 | 3 | 2 | **76132167** | 79 | 0 | 1533 | 25153 |
| **Fastest** | 3 | 3 | 4 | 1103696 | **1422** | 0 | 2778 | 38398 |

Table 5.2: Conv3D IP cycle unrolling cycles exploration.

| *Bytes* | **Conv3D_0** | **Conv3D_1** | **Conv3D_2** | **Conv3D_3** | **Conv3D_4** | **Conv3D_5** | |
|---|---|---|---|---|---|---|---|
| **Bias** | 128 | 128 | 128 | 128 | 128 | 128 | |
| **Kernel** | 18432 | 18432 | 18432 | 18432 | 18432 | 18432 | |
| **Scale** | 32 | 32 | 32 | 32 | 32 | 32 | |
| **Total** | 18592 | 18592 | 18592 | 18592 | 18592 | 18592 | 111552 |

Table 5.1: Bytes used by Conv3D weights.


Forward GRU
F0 F1 F2 F3 F4 F5
Backward GRU
B0 B1 B2 B3 B4 B5
Bidirectional GRU
F0 F1 B0 B1 F2 F3 B2 B3 F4 F5 B4 B5


BGRU
AXI Lite
AXI Stream
Configuration
Input buffer [ 431*64 ]
Bidirectional GRU
2
State Clear
Lines iteration
Cell iteration
Input Line
Output
Output buffer [64/8]
3
1
State Synchronize
SIGMOID lookup table [ 256 ]
TANH lookup table [ 256 ]
SIGMOID
TANH
GRU Cell
State [ 1 ]
New State [ 1 ]
Output AXI Stream
Weights
F Bias[3]
F RBias[3]
F. Kernel [ 64*3*1 ]
F RKernel [ 1*3*1 ]
B Bias[3]
B RBias[3]
B Kernel [ 64*3*1 ]
B RKernel [ 1*3*1 ]
F Bias[3]
F RBias[3]
F. Kernel [ 2*3*1 ]
F RKernel [ 1*3*1 ]
B Bias[3]
B RBias[3]
B Kernel [ 2*3*1 ]
B RKernel [ 1*3*1 ]

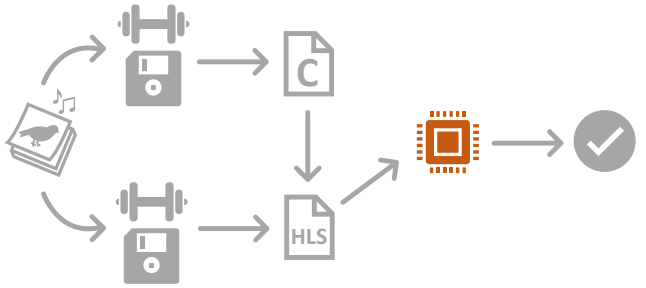
| *Conv3D Unroll* | **Cycles** | **BRAM** | **DSP** | **FF** | **LUT** |
|---|---|---|---|---|---|
| **Ultra96-V2** | - | 216 | 360 | 141120 | 70560 |
| **BGRU IP** | 186633 | 22 | 0 | 1438 | 10045 |

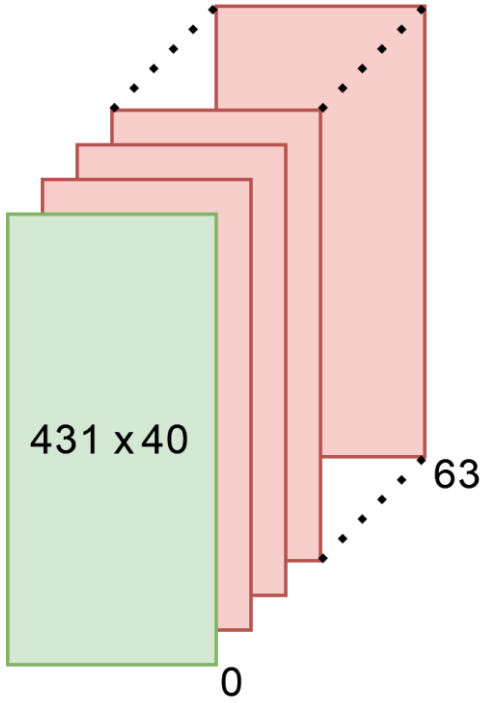
Table 5.4: BGRU IP expected FPGA resource usage.

| *Bytes* | **BGRU_0** | **BGRU_1** | |
|---|---|---|---|
| **Forward Bias** | 3 | 3 | |
| **F. Recurrent Bias** | 3 | 3 | |
| **F. Kernel** | 192 | 6 | |
| **F. Recurrent Kernel** | 3 | 3 | |
| **Backward Bias** | 3 | 3 | |
| **B. Recurrent Bias** | 3 | 3 | |
| **B. Kernel** | 192 | 6 | |
| **B. Recurrent Kernel** | 3 | 3 | |
| **Total** | 402 | 30 | 432 |

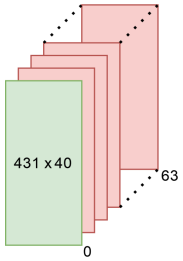
Table 5.3: Bytes used by BGRU weights.

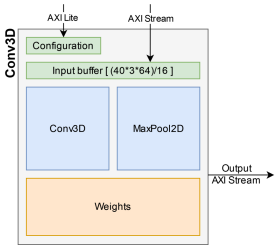
# Blocos IP e Auxiliares

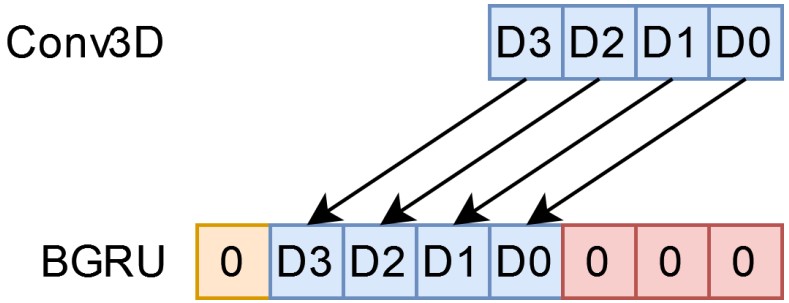


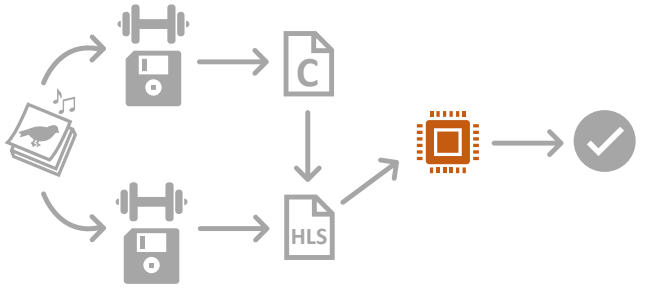



# Blocos IP e Auxiliares

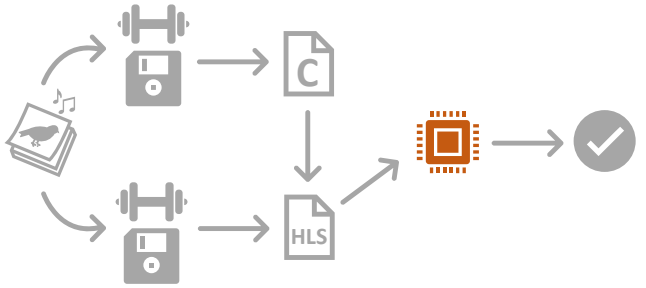


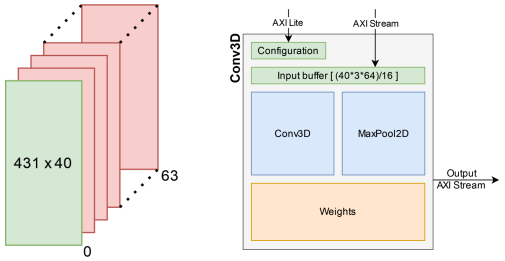



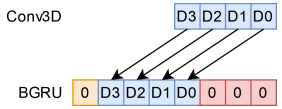



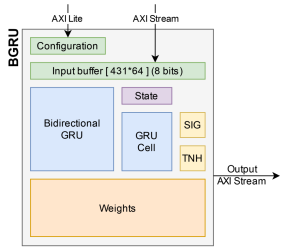



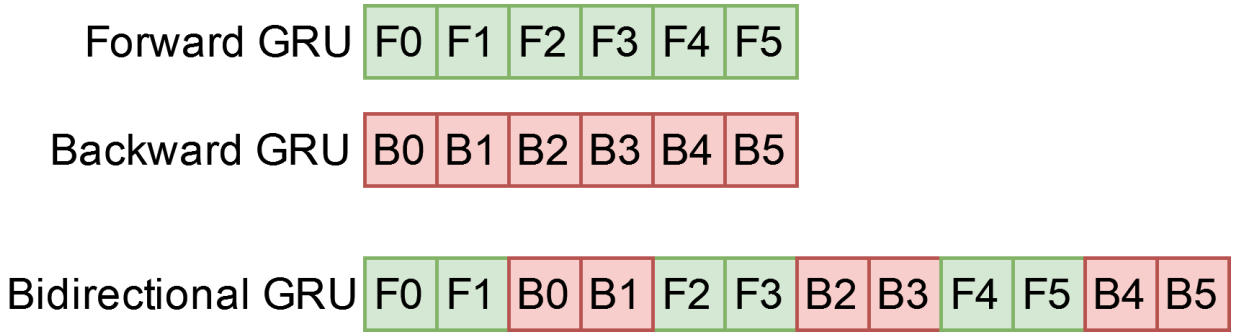
# Blocos IP e Auxiliares



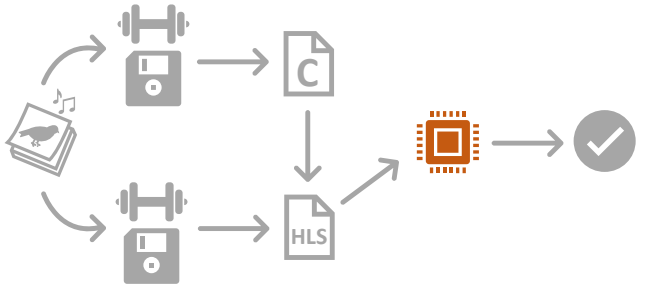



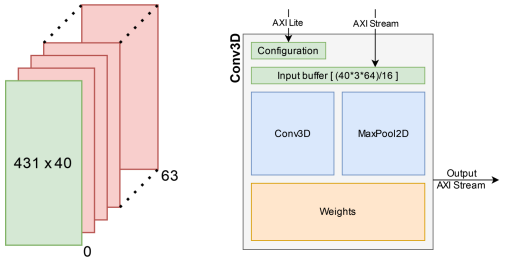



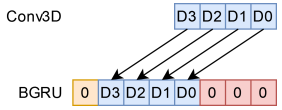



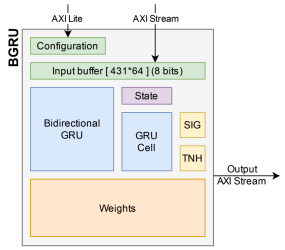



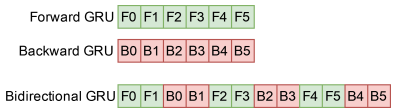
# Blocos IP e Auxiliares

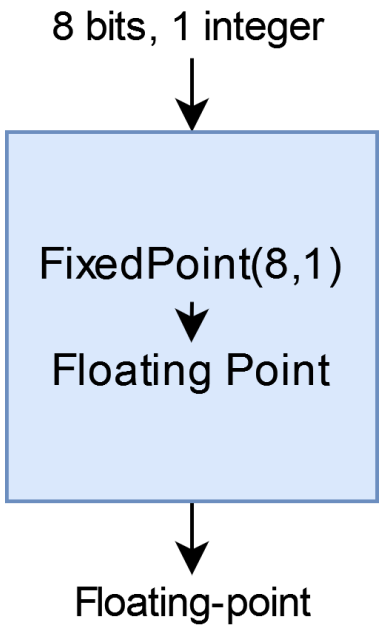


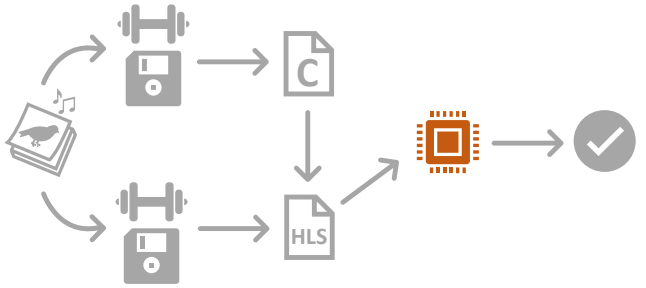



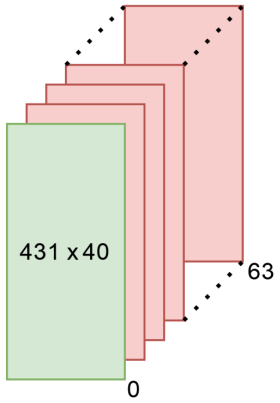



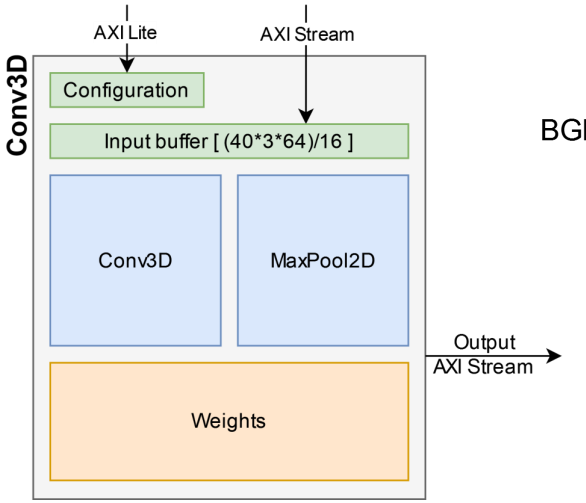



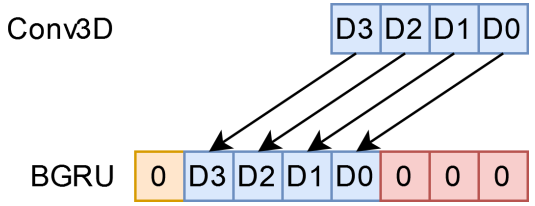



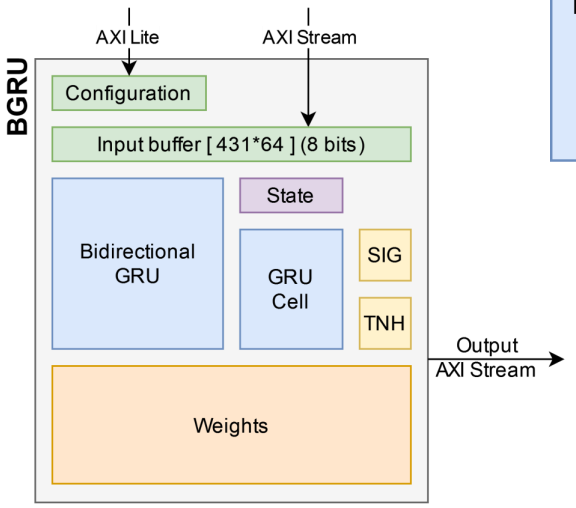
# Blocos IP e Auxiliares

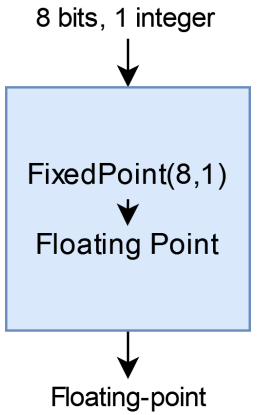


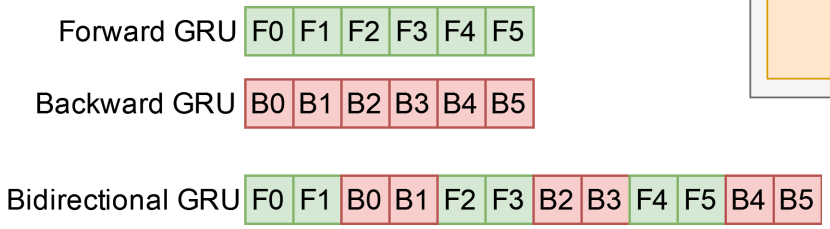



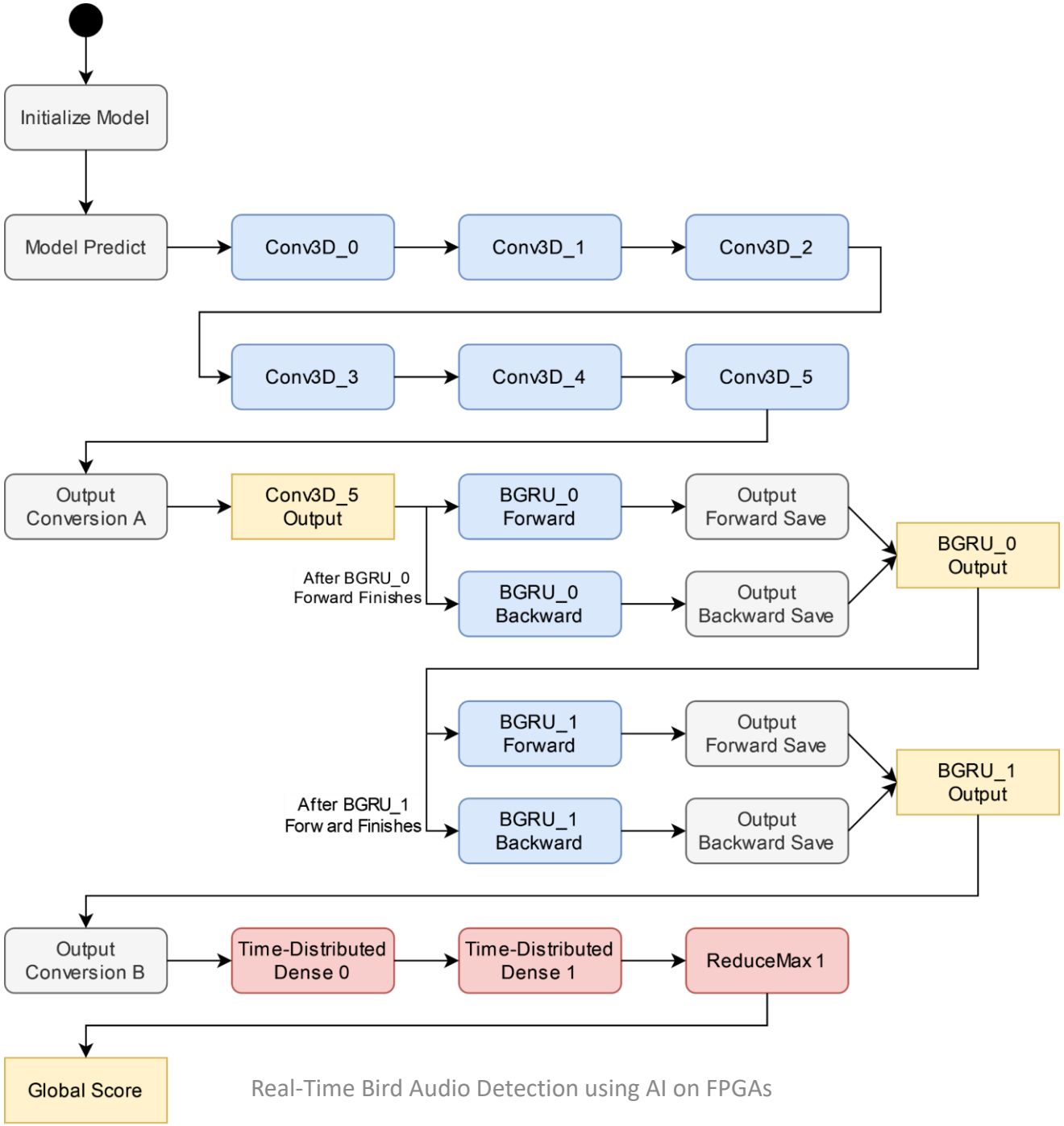



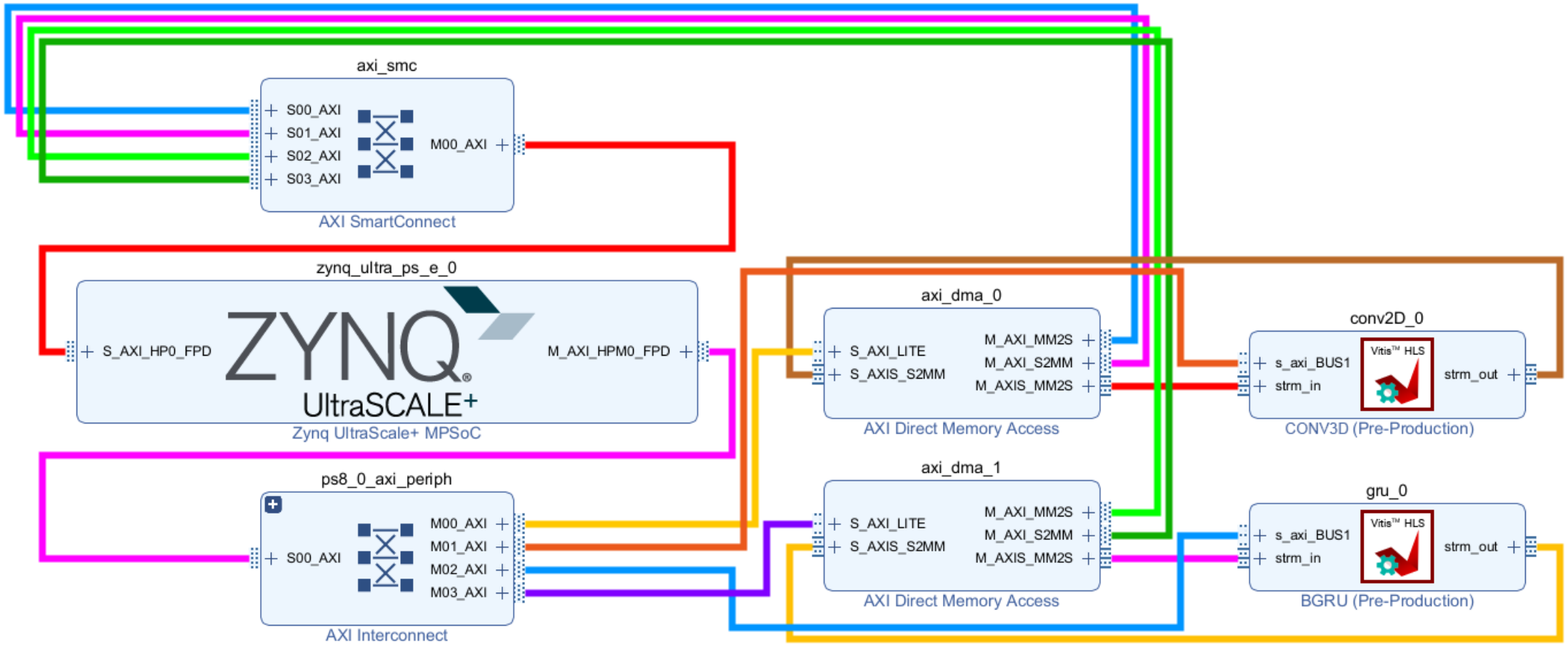




Initialize Model
Model Predict
Conv3D_0
Conv3D_1
Conv3D_2
Conv3D_3
Conv3D_4
Conv3D_5
Output Conversion A
Conv3D_5 Output
BGRU_0 Forward
Output Forward Save
After BGRU_0 Forward Finishes
BGRU_0 Backward
Output Backward Save
BGRU_0 Output
BGRU_1 Forward
Output Forward Save
After BGRU_1 Forw ard Finishes
BGRU_1 Backward
Output Backward Save
BGRU_1 Output
Output Conversion B
Time-Distributed Dense 0
Time-Distributed Dense 1
ReduceMax 1
Global Score


axi_smc
S00_AXI
S01_AXI
S02_AXI
S03_AXI
M00_AXI
AXI SmartConnect
zynq_ultra_ps_e_0
S_AXI_HP0_FPD
ZYNQ
UltraSCALE+
M_AXI_HPM0_FPD
Zynq UltraScale+ MPSoC
ps8_0_axi_periph
S00_AXI
M00_AXI
M01_AXI
M02_AXI
M03_AXI
AXI Interconnect
axi_dma_0
S_AXI_LITE
S_AXIS_S2MM
M_AXI_MM2S
M_AXI_S2MM
M_AXIS_MM2S
AXI Direct Memory Access
axi_dma_1
S_AXI_LITE
S_AXIS_S2MM
M_AXI_MM2S
M_AXI_S2MM
M_AXIS_MM2S
AXI Direct Memory Access
conv2D_0
s_axi_BUS1
strm_in
Vitis™ HLS
strm_out
CONV3D (Pre-Production)
gru_0
s_axi_BUS1
strm_in
Vitis™ HLS
strm_out
BGRU (Pre-Production)

| | *Count* | | *Percentage* | |
|---|---|---|---|---|
| | **Correct** | **Incorrect** | **Correct** | **Incorrect** |
| **Original** | 359 | 41 | 89.75% | 10.25% |
| **Modified** | 345 | 55 | 86.25% | 13.75% |
| **QKeras** | 320 | 80 | 80.00% | 20.00% |
| **FPGA** | 318 | 82 | 79.50% | 20.50% |

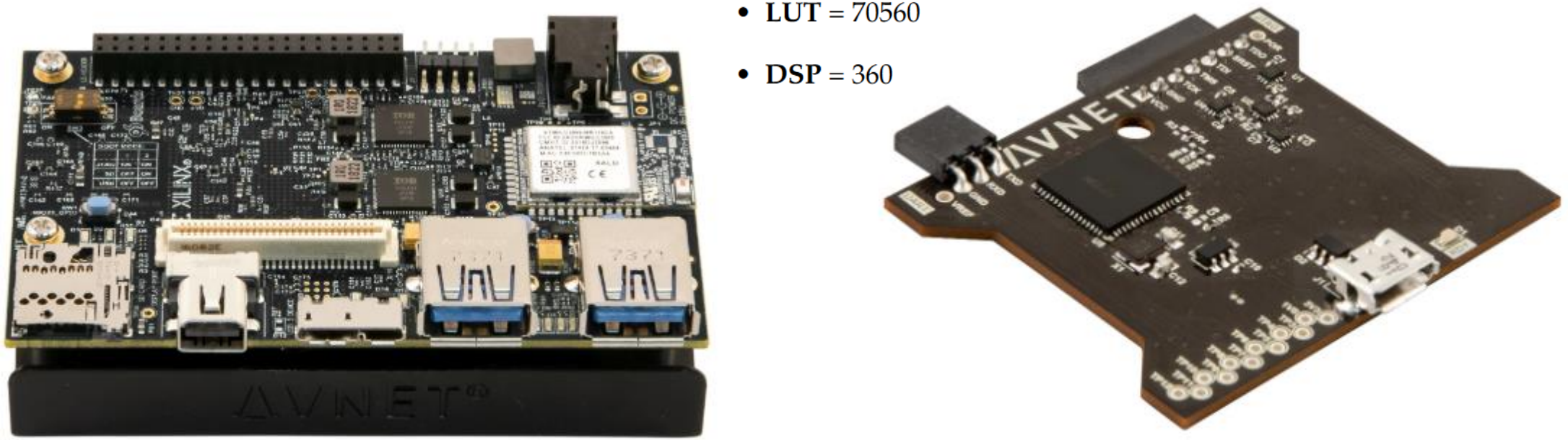
Table 6.1: Accuracy between the Original, Modified, QKeras, and FPGA models.

| *Weights memory* | **Convolutions** | **1st GRU** | **2nd GRU** | **Non-Quantized** | **Total** |
|---|---|---|---|---|---|
| **Model w/ 64 cells** | 111552 B | 49920 B | 74496 B | 66052 B | 302020 B |
| **Model w/ 1 cell** | 111552 B | 402 B | 30 B | 66052 B | 178036 B |
| **Reduction Factor** | 1 | 124.2 | 2483.2 | 1 | 1.7 |

Table 6.2: Weights memory usage comparison between a model with 64 GRU cells and 1 GRU cell, using the previously mentioned weights calculations.

UltraScale+ ZU3CG[3] MPSoC

- **BRAM** = 216
- **FF** = 141120
- **LUT** = 70560
- **DSP** = 360



(a) Avnet Ultra96-V2

(b) AES-ACC-U96-JTAG

Figure 6.1: Target Platform

| *Milliseconds* | **Conv 0** | **Conv 1** | **Conv 2** | **Conv 3** | **Conv 4** | **Conv 5** | **Total** |
|---|---|---|---|---|---|---|---|
| **Baseline** | 497.2 | 502.0 | 248.6 | 251.0 | 124.3 | 124.4 | 1635.5 |
| **Optimized** | 188.3 | 193.1 | 94.1 | 96.6 | 47.1 | 47.2 | 666.4 |
| **Speed Up Factor** | 2.64 | 2.60 | 2.64 | 2.60 | 2.64 | 2.64 | 2.63 |

Table 6.3: Comparison of the milliseconds used in the CNN portion of the model, between Baseline and Optimized.

| *FPGA Resources* | **BRAM** | **DSP** | **FF** | **LUT** |
|---|---|---|---|---|
| **Baseline** | 44.5 | 0 | 1052 | 1616 |
| **Optimized** | 85 | 0 | 1256 | 3202 |
| **Resources Increment Factor** | 1.9 | 1 | 1.19 | 1.98 |

Table 6.4: FPGA resources comparison used in the CNN portion of the model, between baseline and optimized.

| *Milliseconds* | **BGRU 0** | **BGRU 1** | **Total** |
|---|---|---|---|
| **Optimized** | 4.208 | 0.4396 | 4.646 |

Table 6.5: Results of the execution time of both Bidirectional GRUs.

| *FPGA Resources* | **BRAM** | **DSP** | **FF** | **LUT** |
|---|---|---|---|---|
| **Optimized** | 10 | 0 | 1374 | 3304 |

Table 6.6: FPGA resources used in the RNN portion of the model.

| *Milliseconds* | **CNN to RNN** | **RNN to T.Dist.** | **Total** |
|---|---|---|---|
| **Optimized** | 0.577 | 0.038 | 0.615 |

Table 6.7: Execution times taken on pre-processing between CNN, RNN, and final layers.

| *Milliseconds* | **Time-Distributed 0** | **Time-Distributed 1** | **Reduce Max** | **Total** |
|---|---|---|---|---|
| **Optimized** | 6.220 | 1.202 | 0.009 | 7.431 |

Table 6.8: Execution times of the final layers running on CPU.

| *Milliseconds* | **CNN** | **CNN to RNN** | **RNN** | **RNN to T.D.** | **CPU Layers** | **Total** |
|---|---|---|---|---|---|---|
| **Baseline** | 1635.5 | 0.577 | 4.646 | 0.038 | 7.431 | 1648.2 |
| **Optimized** | 666.4 | 0.577 | 4.646 | 0.038 | 7.431 | 679.1 |
| **Speed Up Factor** | 162.9% | 100% | 100% | 100% | 100% | 159.2% |

Table 6.9: Comparison of the number of milliseconds used in different sections of the model, between Baseline and Optimized.

| | CNN | CNN to RNN | RNN | RNN to T.D. | CPU Layers | Total |
|---|---|---|---|---|---|---|
| **Optimized** | 666.4 ms | 0.577 ms | 4.646 ms | 0.038 ms | 7.431 ms | 679.1 ms |
| **% Time Executing** | 98.13 % | 0.08 % | 0.68 % | 0.006 % | 1.09 % | 100 % |

Table 6.10: Presenting the percentage of execution time across the multiple sections of the Optimized implementation.

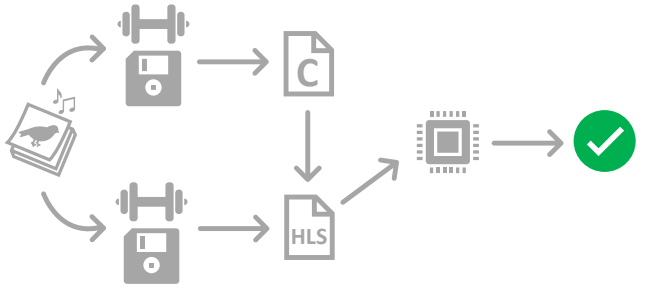
| *FPGA Resources* | **BRAM** | **DSP** | **FF** | **LUT** |
|---|---|---|---|---|
| **Conv3D IP** | 85 | 0 | 1256 | 3202 |
| **BGRU IP** | 10 | 0 | 1374 | 3304 |
| **Optimized** | 101 | 0 | 13971 | 14366 |
| **ZU3CG FPGA** | 216 | 360 | 141120 | 70560 |
| **Usage %** | 46% | 0% | 9.9% | 20.36% |

Table 6.11: Optimized resource usage.

| *Milliseconds* | **CNN** | **CNN to RNN** | **RNN** | **RNN to T.D.** | **CPU Layers** | **Total** |
|---|---|---|---|---|---|---|
| **ARM** | 53477.6 | 0 | 10.52 | 0 | 7.431 | 53495.3 |
| **Optimized** | 666.4 | 0.577 | 4.646 | 0.038 | 7.431 | 679.1 |
| **Speed Up Factor** | 8025% | 0% | 226% | 0% | 100% | 7877% |

Table 6.12: Comparison of the number of milliseconds used in different sections of the model, between ARM CPU and Optimized.

# Resultados e Desempenho




- Original microfaune_ai model without any modification, using float.
- Modified model that only uses 1 GRU cell, without quantization, using float.
- Quantized modified model using QKeras. Note that only the CNN part of the model is quantized at 4 bits with 0 integers, while the rest uses float.
- Quantized modified model, with the RNN portion being quantized with truncation 8 bits with 1 integer, running on the FPGA.

| | *Count* | | *Percentage* | |
|---|---|---|---|---|
| | **Correct** | **Incorrect** | **Correct** | **Incorrect** |
| **Original** | 359 | 41 | 89.75% | 10.25% |
| **Modified** | 345 | 55 | 86.25% | 13.75% |
| **QKeras** | 320 | 80 | 80.00% | 20.00% |
| **FPGA** | 318 | 82 | 79.50% | 20.50% |

Table 6.1: Accuracy between the Original, Modified, QKeras, and FPGA models.

| *Weights memory* | **Convolutions** | **1st GRU** | **2nd GRU** | **Non-Quantized** | **Total** |
|---|---|---|---|---|---|
| **Model w/ 64 cells** | 111552 B | 49920 B | 74496 B | 66052 B | 302020 B |
| **Model w/ 1 cell** | 111552 B | 402 B | 30 B | 66052 B | 178036 B |
| **Reduction Factor** | 1 | 124.2 | 2483.2 | 1 | 1.7 |

Table 6.2: Weights memory usage comparison between a model with 64 GRU cells and 1 GRU cell, using the previously mentioned weights calculations.

| *Milliseconds* | **CNN** | **CNN to RNN** | **RNN** | **RNN to T.D.** | **CPU Layers** | **Total** |
|---|---|---|---|---|---|---|
| **Baseline** | 1635.5 | 0.577 | 4.646 | 0.038 | 7.431 | 1648.2 |
| **Optimized** | 666.4 | 0.577 | 4.646 | 0.038 | 7.431 | 679.1 |
| **Speed Up Factor** | 162.9% | 100% | 100% | 100% | 100% | 159.2% |

Table 6.9: Comparison of the number of milliseconds used in different sections of the model, between Baseline and Optimized.

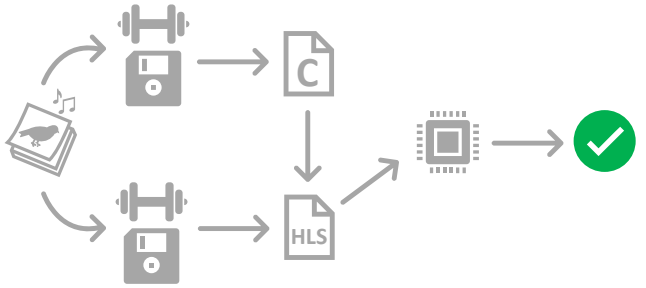
| *FPGA Resources* | **BRAM** | **DSP** | **FF** | **LUT** |
|---|---|---|---|---|
| **Conv3D IP** | 85 | 0 | 1256 | 3202 |
| **BGRU IP** | 10 | 0 | 1374 | 3304 |
| **Optimized** | 101 | 0 | 13971 | 14366 |
| **ZU3CG FPGA** | 216 | 360 | 141120 | 70560 |
| **Usage %** | 46% | 0% | 9.9% | 20.36% |

Table 6.11: Optimized resource usage.

| *Milliseconds* | **CNN** | **CNN to RNN** | **RNN** | **RNN to T.D.** | **CPU Layers** | **Total** |
|---|---|---|---|---|---|---|
| **ARM** | 53477.6 | 0 | 10.52 | 0 | 7.431 | 53495.3 |
| **Optimized** | 666.4 | 0.577 | 4.646 | 0.038 | 7.431 | 679.1 |
| **Speed Up Factor** | 8025% | 0% | 226% | 0% | 100% | 7877% |

Table 6.12: Comparison of the number of milliseconds used in different sections of the model, between ARM CPU and Optimized.

# Desempenho e Recursos




| *ms* | CNN | RNN | Software | Total |
|---|---|---|---|---|
| Baseline | 1635 | 4.65 | 7.4 | 1648 |
| **Optimized** | **666** | **4.65** | **7.4** | **679** |
| Speed Up | 163% | 100% | 100% | 159.2% |

Comparação de desempenho Baseline vs Optimizado

| *bytes* | Total |
|---|---|
| Model w/ 64 cells | 302 020 B |
| **Model w/ 1 cell** | **178 036 B** |
| Reduction Factor | 169.6% |

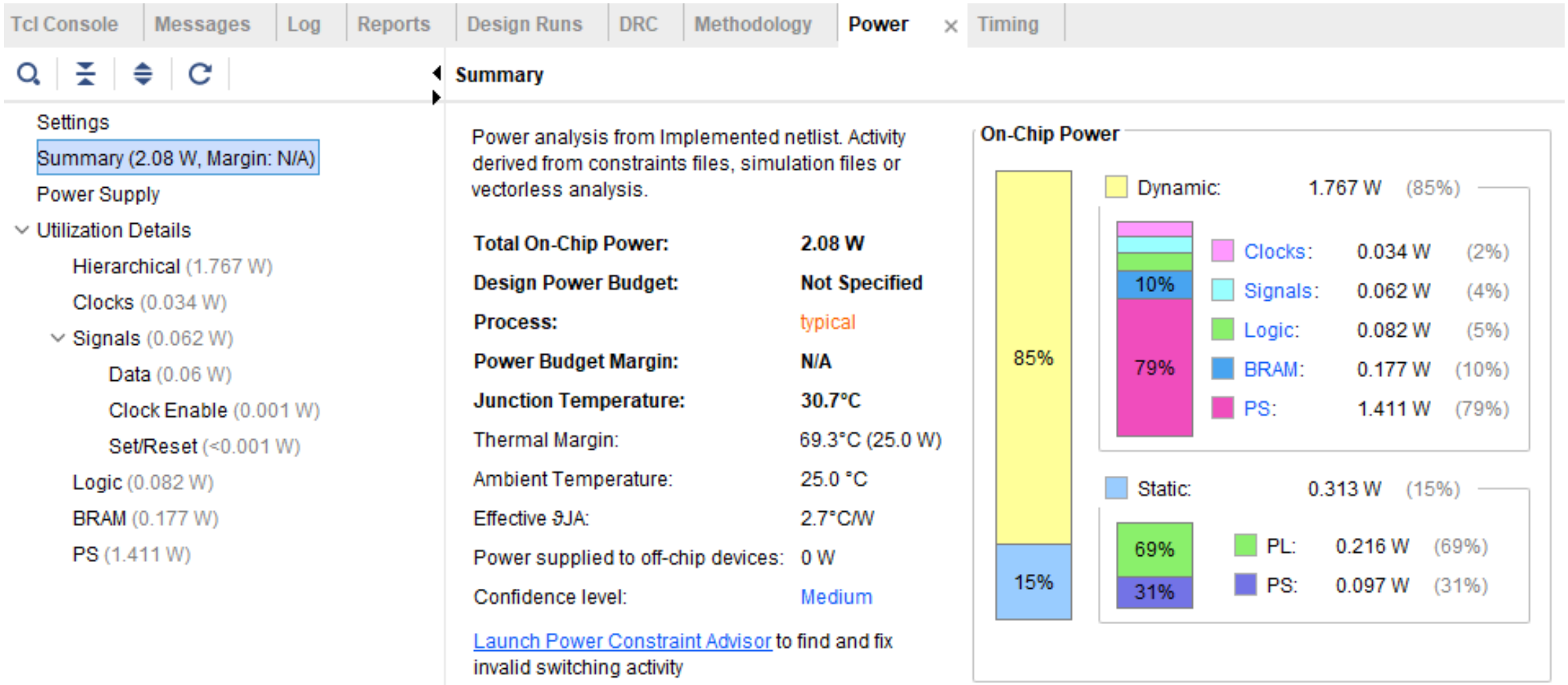
Comparação de recursos 64 células GRU vs 1 célula

| *ms* | CNN | RNN | Software | Total |
|---|---|---|---|---|
| ARM | 53477 | 10.52 | 7.4 | 53495 |
| **Optimized** | **666** | **4.65** | **7.4** | **679** |
| Speed Up | 8025% | 226% | 100% | 7877% |

Comparação de desempenho Software vs Optimizado

| *FPGA* | BRAM | DSP | FF | LUT |
|---|---|---|---|---|
| **Optimized** | **101** | **0** | **13971** | **14366** |
| ZU3CG | 216 | 360 | 141120 | 70560 |
| **Usage %** | **46%** | **0%** | **9.9%** | **20.36%** |

Recursos da FPGA usados


Tcl Console
Messages
Log
Reports
Design Runs
DRC
Methodology
Power
Timing
Summary
Settings
Summary (2.08 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (1.767 W)
Clocks (0.034 W)
Signals (0.062 W)
Data (0.06 W)
Clock Enable (0.001 W)
Set/Reset (<0.001 W)
Logic (0.082 W)
BRAM (0.177 W)
PS (1.411 W)
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.
Total On-Chip Power: 2.08 W
Design Power Budget: Not Specified
Process: typical
Power Budget Margin: N/A
Junction Temperature: 30.7°C
Thermal Margin: 69.3°C (25.0 W)
Ambient Temperature: 25.0 °C
Effective ϑJA: 2.7°C/W
Power supplied to off-chip devices: 0 W
Confidence level: Medium
Launch Power Constraint Advisor to find and fix invalid switching activity
On-Chip Power
85%
15%
Dynamic: 1.767 W (85%)
10%
79%
Clocks: 0.034 W (2%)
Signals: 0.062 W (4%)
Logic: 0.082 W (5%)
BRAM: 0.177 W (10%)
PS: 1.411 W (79%)
Static: 0.313 W (15%)
69%
31%
PL: 0.216 W (69%)
PS: 0.097 W (31%)