Application Note

MMA 错误注入

Manojna Manojna

摘要

故障注入和异常处理对于安全系统至关重要，本文档介绍了如何依次注入 MMA 故障。

内容

1 简介.....2
2 指令.....2
2.1 范围.....2
2.2 HWA 指令定义.....2
2.3 故障注入程序.....3
3 流程图.....13
3.1 代码更改.....14
4 总结.....16
5 参考资料.....16

商标

1 简介

矩阵乘法加速器 (MMA) 是一款紧密耦合的矩阵乘法协处理器，可扩展 C7x 处理器架构的标量和矢量功能。通过流引擎 (SE) 馈送，MMA 提供了大量乘法累加 (MAC) 运算、矩阵优化存储和矩阵优化数据移动，这些移动可有效用于密集线性代数运算，而这些运算主导了视觉（CNN、SfM、滤波等）、语音 (xNN)、音频（卷积）、雷达 (FFT) 和状态强化应用中的计算。

本文档介绍了故障注入的顺序过程，尤其重点介绍了矩阵乘法器加速器 (MMA) 诊断。

本文讲解档如何按顺序执行这些测试，以便详细了解故障处理和恢复。

本文档并未讨论在完成错误注入序列之后恢复异常以执行代码的常规流程。有关集成示例，请参阅 SDK 11.2。

2 指令

2.1 范围

本文档仅重点介绍 MMA 的顺序故障注入测试，并详细介绍了如何触发和执行这些故障。

本文档并未讨论顺序测试完成后重置 c7x 的程序。测试后恢复、系统复位和系统状态恢复不在本文档讨论范围内，必须根据系统设计和容错单独进行。

2.2 HWA 指令定义

硬件加速器 (HWA) 同 C7x CPU 紧密耦合。HWA 接受命令并从 C7x CPU 运行，执行特定的计算任务，然后将结果返回给 C7x CPU。

- HWAOPEN：将计算模板发送到 HWA，以便将用于后续 HWA 操作的操作数和计算类型与 HWA 进行通信。
- HWALDA、HWALDB、HWALDC、HWALDAB、HWALDBC：这些各种 HWALD 指令用于从 GRF、LRFL 及 SE0、SE1 寄存器读取值，并将寄存器发送到 HWA 以初始化 HWA 操作数矩阵。
- HWAOP：HWAOP 指令指示 HWA 继续已编程的计算。
- HWAXFER：HWAXFER 指令将计算结果从 HWA 逻辑传输到内部缓冲器。
- HWARCVS：HWARCVS 指令会将存储在 HWA 内部缓冲器中的值传输到 C7x CPU 通用寄存器。
- HWACLOSE：HWACLOSE 关闭 HWA 操作。HWA 中的所有中间值都会被丢弃。

MMA 规范文档中有更多详细信息。请参阅 *C71x DSP CPU、指令集和矩阵乘法加速器*，了解更多详细信息。

2.3 故障注入程序

故障注入程序

为了评估 MMA 诊断，必须在监测结果行为的同时故意将故障注入系统。当 HWA_STATUS 错误代码字段包含非零值时，执行 HWARCV_(0) 指令会触发 c7x 异常。

发生异常后，必须读取 IERR 和 IESR 寄存器以确定故障性质：

- IERR：通过特定标志指示内部异常原因。
- IESR：提供了有关故障类型及子类型的更多详细信息。

2.3.1 方框图

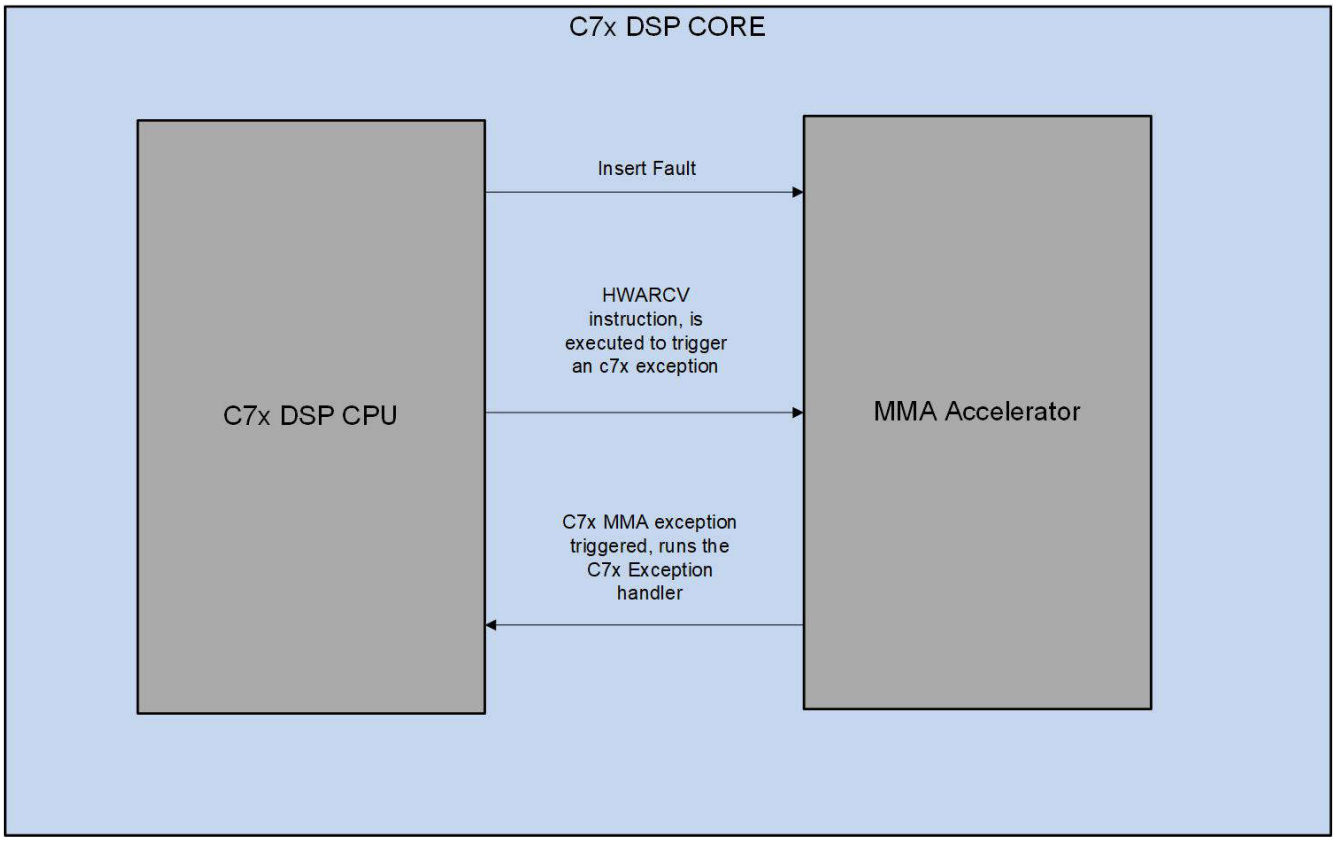

图 2-1. C7x - MMA 错误注入方框图

2.3.2 故障注入：步骤

2.3.2.1 下溢错误

尝试读取传输缓冲器时会导致下溢错误，但 FIFO 中无有效数据（编程错误），以下是注入下溢错误的步骤。

1. HWAOPEN 指令用于写入 HWA_CONFIG 及 HWA_OFFSET。
2. 执行 HWAXFER 以将 HWA_STATUS 的内容传输至内部缓冲器。
3. 执行 HWARCV_(0) 指令，将存储在 HWA 内部缓冲器中的值传输至 C7x CPU 通用寄存器。
4. 重复步骤 3，直到 c7x CPU 的内部缓冲器中没有会触发异常的有效数据。

```
void underflow_exception(void)
{
  __HWA_CONFIG_REG_v1 mma_config_reg;
  mma_config_reg = __gen_HWA_CONFIG_REG_v1();
  __HWA_OFFSET_REG offset_reg;
  offset_reg = __gen_HWA_OFFSET_REG();
  __HWAOPEN(mma_config_reg,offset_reg,__MMA_OPEN_FSM_RESET);
  __HWAADV();
  __HWAADV();
  __HWAXFER(__MMA_XFER_SRC_HWA_STATUS);
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWARCV(0);
  __HWARCV(0);
}
```

2.3.2.2 溢出错误

当尝试读取传输缓冲器时，会引起溢出错误，但早期的 HWA 指令会导致 FIFO 溢出（编程错误）。依照列出的步骤注入错误。

1. HWAOPEN 指令用于写入 HWA_CONFIG 及 HWA_OFFSET。
2. 执行 HWAXFER 以将 HWA_STATUS 的内容传输至内部缓冲器。
3. 重复步骤 2，直到缓冲器溢出。
4. 执行 HWARCV 指令以观察错误。

```
void overflow_exception(void)
{
  __HWA_CONFIG_REG_v1 mma_config_reg;
  mma_config_reg = __gen_HWA_CONFIG_REG_v1();
  __HWA_OFFSET_REG offset_reg;
  offset_reg = __gen_HWA_OFFSET_REG();
  __HWAOPEN(mma_config_reg,offset_reg,__MMA_OPEN_FSM_RESET);
  int loop_begin = 1;
  int loop_end  = 30;
  for(;loop_begin<=loop_end;loop_begin++)
  {
    __HWAXFER(__MMA_XFER_SRC_HWA_STATUS);
    __HWAADV();
    __HWAADV();
    __HWAADV();
    __HWAADV();
  }
  __HWARCV(0);
}
```

2.3.2.3 偏移奇偶校验错误

当在 HWA_OFFSET 寄存器中检测到奇偶校验不匹配时，会发生偏移奇偶校验错误。依照以下步骤注入 HWA_OFFSET 奇偶校验错误。

1. HWAOPEN 指令用于在启用奇偶校验计算并启用奇偶校验（PARITYCTRL = NORMAL）的情况下，写入 HWA_CONFIG 和 HWA_OFFSET。
2. 在要进行测试的特定字段，软件会损坏向量寄存器中的数据值。
3. 在禁用奇偶校验计算并启用奇偶校验 (PARITYCTRL = PNCMCK) 的情况下，HWAOPEN 指令再次用于使用来自该 C7x 向量寄存器的损坏数据写入 HWA_CONFIG 和 HWA_OFFSET。在此模式下，将根据受影响寄存器中使用时计算出的奇偶校验值，检查最初计算并保存的奇偶校验值。
4. 执行 HWAXFER 以传输 HWA_CONFIG/HWA_OFFSET 等内容来传输缓冲器。
5. 执行 HWARCV 指令以观察错误。

```
 __HWA_OFFSET_REG get_corrupted_offset(void)
{
__HWA_OFFSET_REG mma_offset_reg = __gen_HWA_OFFSET_REG();
mma_offset_reg.A_LUT_VAL_1 = 0x01;
return mma_offset_reg; //return the corrupted offset
}
#pragma FUNC_CANNOT_INLINE(offset_parity_test)
#pragma FUNCTION_OPTIONS(offset_parity_test,"--opt_level=0")
void offset_parity_test(void)
{

__HWA_CONFIG_REG_v1 mma_config_reg;
mma_config_reg = __gen_HWA_CONFIG_REG_v1();
mma_config_reg.A_ALUTEN = __MMA_A_CONFIG_LUT2;
mma_config_reg.PARITYCTRL = __MMA_NORMAL;
__HWA_OFFSET_REG offset_reg;
offset_reg = __gen_HWA_OFFSET_REG();
__HWAOPEN(mma_config_reg,offset_reg,__MMA_OPEN_FSM_RESET);
__HWAADV();
__HWAADV();
__HWAADV();
__HWAADV();
__HWA_CONFIG_REG_v1 mma_config_1;
mma_config_1 = __gen_HWA_CONFIG_REG_v1();
mma_config_1.A_ALUTEN = __MMA_A_CONFIG_LUT2;
//Parity computation disabled and parity checking enabled.
mma_config_1.PARITYCTRL = __MMA_PNCM_CK;
__HWA_OFFSET_REG offset_reg2 = get_corrupted_offset(); //This gets the corrupted offset
__HWAADV();
__HWAADV();
__HWAADV();
__HWAADV();
__HWAOPEN(mma_config_1,offset_reg2,__MMA_OPEN_FSM_RESET);
__HWAADV();
__HWAADV();
__HWAADV();
__HWAADV();
__HWAXFER(__MMA_XFER_SRC_HWA_OFFSET);
__HWAADV();
__HWAADV();
__HWAADV();
__HWARCV(0);
__asm(" NOP");
__asm(" NOP");


}
```

2.3.2.4 配置奇偶校验错误

当在 HWA_CONFIG 寄存器中检测到奇偶校验不匹配时，会发生配置奇偶校验错误。以下步骤列出了有关如何注入 HWA_CONFIG 奇偶校验错误的程序。所有 FSM 配置（A、B、C、X FSM 配置）均受奇偶校验保护。

1. HWAOPEN 指令用于在启用奇偶校验计算并启用奇偶校验（PARITYCTRL = NORMAL）的情况下，写入 HWA_CONFIG 和 HWA_OFFSET。
2. 在要进行测试的特定字段，软件会损坏向量寄存器中的数据值。
3. 在禁用奇偶校验计算并启用奇偶校验 (PARITYCTRL = PNCMCK) 的情况下，HWAOPEN 指令再次用于使用来自该 C7x 向量寄存器的损坏数据写入 HWA_CONFIG 和 HWA_OFFSET。在此模式下，将根据受影响寄存器中使用时计算出的奇偶校验值，检查最初计算并保存的奇偶校验值。
4. 执行 HWAXFER 以传输 HWA_CONFIG/HWA_OFFSET 等内容来传输缓冲器。
5. 执行 HWARCV 指令以观察错误

2.3.2.4.1 FSM 配置奇偶校验错误

当在 HWA_CONFIG 寄存器中检测到奇偶校验不匹配时，会发生 A FSM 配置奇偶校验错误。以下步骤列出了有关如何注入 A FSM 配置奇偶校验错误的程序。

1. HWAOPEN 指令用于在启用奇偶校验计算和启用奇偶校验 (PARITYCTRL = NORMAL) 的情况下，写入 HWA_CONFIG 和 HWA_OFFSET，也可使用 Get_MMAconfig() 中提到的参数来配置 HWA_CONFIG。
2. 在要进行测试的特定字段 (A FSM)，软件会损坏向量寄存器中的数据值。
3. 在禁用奇偶校验计算并启用奇偶校验 (PARITYCTRL = PNCMCK) 的情况下，HWAOPEN 指令再次用于使用来自该 C7x 向量寄存器的损坏数据写入 HWA_CONFIG 和 HWA_OFFSET。在此模式下，将根据受影响寄存器中使用时计算出的奇偶校验值，检查最初计算并保存的奇偶校验值。
4. 执行 HWALDA /HWAXFER 以注入错误。错误会在 HWA_STATUS 寄存器中更新。
5. 执行 HWARCV 指令以观察错误

```
#pragma FUNCTION_OPTIONS(Get_AFSM_Corrupt_config,"--opt_level=off")
__HWA_CONFIG_REG_v1 Get_AFSM_Corrupt_config(void)
{
  __HWA_CONFIG_REG_v1 corrupted_config = __gen_HWA_CONFIG_REG_v1();
  corrupted_config.A_ATYPE = __MMA_A_CONFIG_ATYPE_UINT32;
  corrupted_config.A_ALUTEN = __MMA_A_LUTEN_LAST;
  return corrupted_config;
}
#pragma FUNCTION_OPTIONS(Get_MMAconfig,"--opt_level=off")
__HWA_CONFIG_REG_v1 Get_MMAconfig(void)
{
  __HWA_CONFIG_REG_v1 mma_config_reg;
  mma_config_reg = __gen_HWA_CONFIG_REG_v1();
  mma_config_reg.A_ATYPE = __MMA_A_CONFIG_ATYPE_UINT16;
  mma_config_reg.A_ALUTEN = __MMA_A_LUTEN_LAST;
  mma_config_reg.B_BSWPER = 1000;
  mma_config_reg.B_BTYPE = __MMA_B_CONFIG_SIZE16;
  mma_config_reg.B_BSTART = 1;
  mma_config_reg.C_ATYPE = __MMA_C_CONFIG_ATYPE_SA;
  mma_config_reg.C_BTYPE = __MMA_C_CONFIG_BTYPE_UINT16;
  mma_config_reg.PARITYCTRL = __MMA_NORMAL;
  mma_config_reg.C_BSTART = 1; /* Initial B bank selection for reading B matrix data for the matrix
computations */
  mma_config_reg.C_CRSTART = 1; /* Initial C bank selection for reading operands */
  mma_config_reg.C_CWSTART = 1; /* Initial C bank selection for writing computation results */
  mma_config_reg.C_CLSTART = 1;
  return mma_config_reg;
}
#pragma FUNCTION_OPTIONS(MMA_AFSM_configParityError_injection,"--opt_level=off")
void MMA_AFSM_configParityError_injection(void)
{
  __HWA_CONFIG_REG_v1 mma_config_reg = Get_MMAconfig();
  __HWA_OFFSET_REG offset_reg;
  offset_reg = __gen_HWA_OFFSET_REG();
  __HWAOPEN(mma_config_reg,offset_reg,__MMA_OPEN_FSM_RESET);
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __asm(" HWALDA .L2 VB1");
```

```
  __HWAXFER(__MMA_XFER_SRC_C);
  __HWA_CONFIG_REG_v1 mma_config_corrupt = Get_AFSM_Corrupt_config();
  mma_config_corrupt.PARITYCTRL = __MMA_PNCM_CK;
  __HWA_OFFSET_REG offset_reg2 = __gen_HWA_OFFSET_REG();
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWAOPEN(mma_config_corrupt,offset_reg2,__MMA_OPEN_FSM_RESET);
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __asm(" HWALDA .L2 VB0"); //Load to A matrix
  __HWAXFER(__MMA_XFER_SRC_HWA_CONFIG);
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWARCV(0);
  __asm(" NOP");
  __asm(" NOP");
}
```

2.3.2.4.2 B FSM 配置奇偶校验错误

当在 HWA_CONFIG 寄存器中检测到奇偶校验不匹配时，会发生 B FSM 配置奇偶校验错误。以下步骤列出了有关如何注入 B FSM 配置奇偶校验错误的程序。

1. HWAOPEN 指令用于在启用奇偶校验计算和启用奇偶校验 (PARITYCTRL = NORMAL) 的情况下，写入 HWA_CONFIG 和 HWA_OFFSET，也可使用 Get_MMAconfig() 中提到的参数来配置 HWA_CONFIG。
2. 在要进行测试的特定字段 (B FSM)，软件会损坏向量寄存器中的数据值。
3. 在禁用奇偶校验计算并启用奇偶校验 (PARITYCTRL = PNCMCK) 的情况下，HWAOPEN 指令再次用于使用来自该 C7x 向量寄存器的损坏数据写入 HWA_CONFIG 和 HWA_OFFSET。在此模式下，将根据受影响寄存器中使用时计算出的奇偶校验值，检查最初计算并保存的奇偶校验值。
4. 执行 HWALDB /HWAXFER 以注入错误。错误会在 HWA_STATUS 寄存器中更新。
5. 执行 HWARCV 指令以观察错误

```
#pragma FUNCTION_OPTIONS(Get_BFSM_Corrupt_config,"--opt_level=off")
__HWA_CONFIG_REG_v1 Get_BFSM_Corrupt_config(void)
{
  __HWA_CONFIG_REG_v1 corrupted_config = __gen_HWA_CONFIG_REG_v1();
  corrupted_config.B_BSWPER = 100;
  corrupted_config.B_BTYPE = __MMA_B_CONFIG_SIZE32;
  return corrupted_config;
}
#pragma FUNCTION_OPTIONS(Get_MMAconfig,"--opt_level=off")
__HWA_CONFIG_REG_v1 Get_MMAconfig(void)
{
  __HWA_CONFIG_REG_v1 mma_config_reg;
  mma_config_reg = __gen_HWA_CONFIG_REG_v1();
  mma_config_reg.A_ATYPE = __MMA_A_CONFIG_ATYPE_UINT16;
  mma_config_reg.A_ALUTEN = __MMA_A_LUTEN_LAST;
  mma_config_reg.B_BSWPER = 1000;
  mma_config_reg.B_BTYPE = __MMA_B_CONFIG_SIZE16;
  mma_config_reg.B_BSTART = 1;
  mma_config_reg.C_ATYPE = __MMA_C_CONFIG_ATYPE_SA;
  mma_config_reg.C_BTYPE = __MMA_C_CONFIG_BTYPE_UINT16;
  mma_config_reg.PARITYCTRL = __MMA_NORMAL;
  mma_config_reg.C_BSTART = 1; /* Initial B bank selection for reading B matrix data for the matrix
computations */
  mma_config_reg.C_CRSTART = 1; /* Initial C bank selection for reading operands */
  mma_config_reg.C_CWSTART = 1; /* Initial C bank selection for writing computation results */
  mma_config_reg.C_CLSTART = 1;
  return mma_config_reg;
}
#pragma FUNCTION_OPTIONS(MMA_BFSM_configParityError_injection,"--opt_level=off")
void MMA_BFSM_configParityError_injection(void)
{
__HWA_CONFIG_REG_v1 mma_config_reg = Get_MMAconfig();
  __HWA_OFFSET_REG offset_reg;
  offset_reg = __gen_HWA_OFFSET_REG();
  __HWAOPEN(mma_config_reg,offset_reg,__MMA_OPEN_FSM_RESET);
  __HWAADV();
```

```
  __HWAADV();
  __HWAADV();
  __asm(" HWALDB .L2 VB1");
  __HWAXFER(__MMA_XFER_SRC_C);
  __HWA_CONFIG_REG_v1 mma_config_corrupt = Get_BFSM_Corrupt_config();
  mma_config_corrupt.PARITYCTRL = __MMA_PNCM_CK;
  __HWA_OFFSET_REG offset_reg2 = __gen_HWA_OFFSET_REG();
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWAOPEN(mma_config_corrupt,offset_reg2,__MMA_OPEN_FSM_RESET);
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __asm(" HWALDB .L2 VB0"); //Load to B matrix
  __HWAXFER(__MMA_XFER_SRC_HWA_CONFIG);
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWARCV(0);
  __asm(" NOP");
  __asm(" NOP");

}
```

2.3.2.4.3 C FSM 配置奇偶校验错误

当在 HWA_CONFIG 寄存器中检测到奇偶校验不匹配时，会发生 C FSM 配置奇偶校验错误。以下步骤列出了有关如何注入 C FSM 配置奇偶校验错误的程序。

1. HWAOPEN 指令用于在启用奇偶校验计算和启用奇偶校验 (PARITYCTRL = NORMAL) 的情况下，写入 HWA_CONFIG 和 HWA_OFFSET，也可使用 Get_MMAconfig() 中提到的参数来配置 HWA_CONFIG。
2. 在要进行测试的特定字段 (C FSM)，软件会损坏向量寄存器中的数据值。
3. 在禁用奇偶校验计算并启用奇偶校验 (PARITYCTRL = PNCMCK) 的情况下，HWAOPEN 指令再次用于使用来自该 C7x 向量寄存器的损坏数据写入 HWA_CONFIG 和 HWA_OFFSET。在此模式下，将根据受影响寄存器中使用时计算出的奇偶校验值，检查最初计算并保存的奇偶校验值。
4. 执行 HWALDC /HWAXFER 以注入错误。错误会在 HWA_STATUS 寄存器中更新。
5. 执行 HWARCV 指令以观察错误。

```
#pragma FUNCTION_OPTIONS(Get_CFSM_Corrupt_config,"--opt_level=off")
__HWA_CONFIG_REG_v1 Get_CFSM_Corrupt_config(void)
{
  __HWA_CONFIG_REG_v1 corrupted_config = __gen_HWA_CONFIG_REG_v1();
  corrupted_config.C_ATYPE = __MMA_C_CONFIG_ATYPE_UA;
  corrupted_config.C_BTYPE = __MMA_C_CONFIG_BTYPE_UINT16;
  return corrupted_config;
}
#pragma FUNCTION_OPTIONS(Get_MMAconfig,"--opt_level=off")
__HWA_CONFIG_REG_v1 Get_MMAconfig(void)
{
  __HWA_CONFIG_REG_v1 mma_config_reg;
  mma_config_reg = __gen_HWA_CONFIG_REG_v1();
  mma_config_reg.A_ATYPE = __MMA_A_CONFIG_ATYPE_UINT16;
  mma_config_reg.A_ALUTEN = __MMA_A_LUTEN_LAST;
  mma_config_reg.B_BSWPER = 1000;
  mma_config_reg.B_BTYPE = __MMA_B_CONFIG_SIZE16;
  mma_config_reg.B_BSTART = 1;
  mma_config_reg.C_ATYPE = __MMA_C_CONFIG_ATYPE_SA;
  mma_config_reg.C_BTYPE = __MMA_C_CONFIG_BTYPE_UINT16;
  mma_config_reg.PARITYCTRL = __MMA_NORMAL;
  mma_config_reg.C_BSTART = 1; /* Initial B bank selection for reading B matrix data for the matrix
computations */
  mma_config_reg.C_CRSTART = 1; /* Initial C bank selection for reading operands */
  mma_config_reg.C_CWSTART = 1; /* Initial C bank selection for writing computation results */
  mma_config_reg.C_CLSTART = 1;
  return mma_config_reg;
}

#pragma FUNCTION_OPTIONS(MMA_CFSM_configParityError_injection,"--opt_level=off")
void MMA_CFSM_configParityError_injection(void)
{
```

```
  __HWA_CONFIG_REG_v1 mma_config_reg = Get_MMAconfig();
  __HWA_OFFSET_REG offset_reg;
  offset_reg = __gen_HWA_OFFSET_REG();
  __HWAOPEN(mma_config_reg,offset_reg,__MMA_OPEN_FSM_RESET);
 __HWAADV();
  __HWAADV();
  __HWAADV();
  __asm(" HWALDC .L2 VB1");
  __HWAXFER(__MMA_XFER_SRC_C);
  __HWA_CONFIG_REG_v1 mma_config_corrupt = Get_CFSM_Corrupt_config();
  mma_config_corrupt.PARITYCTRL = __MMA_PNCM_CK;
  __HWA_OFFSET_REG offset_reg2 = __gen_HWA_OFFSET_REG();
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWAOPEN(mma_config_corrupt,offset_reg2,__MMA_OPEN_FSM_RESET);
   __HWAADV();
  __HWAADV();
  __HWAADV();
  __asm(" HWALDC .L2 VB0"); //Load to C Matrix
  __HWAXFER(__MMA_XFER_SRC_HWA_CONFIG);
 __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWARCV(0);
  __asm(" NOP");
  __asm(" NOP");
}
```

2.3.2.4.4 X FSM 配置奇偶校验错误

当在 HWA_CONFIG 寄存器中检测到奇偶校验不匹配时，会发生 X FSM 配置奇偶校验错误。以下步骤列出了有关如何注入 X FSM 配置奇偶校验错误的程序。

1. HWAOPEN 指令用于在启用奇偶校验计算和启用奇偶校验 (PARITYCTRL = NORMAL) 的情况下，写入 HWA_CONFIG 和 HWA_OFFSET，也可使用 Get_MMAconfig() 中提到的参数来配置 HWA_CONFIG。
2. 在要进行测试的特定字段 (X FSM)，软件会损坏向量寄存器中的数据值。
3. 在禁用奇偶校验计算并启用奇偶校验 (PARITYCTRL = PNCMCK) 的情况下，HWAOPEN 指令再次用于使用来自该 C7x 向量寄存器的损坏数据写入 HWA_CONFIG 和 HWA_OFFSET。在此模式下，将根据受影响寄存器中使用时计算出的奇偶校验值，检查最初计算并保存的奇偶校验值。
4. 执行 HWAXFER 以注入错误。错误会在 HWA_STATUS 寄存器中更新。
5. 执行 HWARCV 指令以观察错误。

```
#pragma FUNCTION_OPTIONS(Get_XFSM_Corrupt_config,"--opt_level=off")
__HWA_CONFIG_REG_v1 Get_XFSM_Corrupt_config(void)
{
  __HWA_CONFIG_REG_v1 corrupted_config = __gen_HWA_CONFIG_REG_v1();
  corrupted_config.X_ReLU = 1;
  return corrupted_config;
}
#pragma FUNCTION_OPTIONS(Get_MMAconfig,"--opt_level=off")
__HWA_CONFIG_REG_v1 Get_MMAconfig(void)
{
  __HWA_CONFIG_REG_v1 mma_config_reg;
  mma_config_reg = __gen_HWA_CONFIG_REG_v1();
  mma_config_reg.A_ATYPE = __MMA_A_CONFIG_ATYPE_UINT16;
  mma_config_reg.A_ALUTEN = __MMA_A_LUTEN_LAST;
  mma_config_reg.B_BSWPER = 1000;
  mma_config_reg.B_BTYPE = __MMA_B_CONFIG_SIZE16;
  mma_config_reg.B_BSTART = 1;
  mma_config_reg.C_ATYPE = __MMA_C_CONFIG_ATYPE_SA;
  mma_config_reg.C_BTYPE = __MMA_C_CONFIG_BTYPE_UINT16;
  mma_config_reg.PARITYCTRL = __MMA_NORMAL;
  mma_config_reg.C_BSTART = 1; /* Initial B bank selection for reading B matrix data for the matrix
computations */
  mma_config_reg.C_CRSTART = 1; /* Initial C bank selection for reading operands */
  mma_config_reg.C_CWSTART = 1; /* Initial C bank selection for writing computation results */
  mma_config_reg.C_CLSTART = 1;
  return mma_config_reg;
}
#pragma FUNCTION_OPTIONS(MMA_XFSM_configParityError_injection,"--opt_level=off")
```

```
void MMA_XFSM_configParityError_injection(void)
{
  __HWA_CONFIG_REG_v1 mma_config_reg = Get_MMAconfig();
  __HWA_OFFSET_REG offset_reg;
  offset_reg = __gen_HWA_OFFSET_REG();
  __HWAOPEN(mma_config_reg,offset_reg,__MMA_OPEN_FSM_RESET); //Initialises Config,offset
 __HWAXFER(__MMA_XFER_SRC_HWA_CONFIG);
  __HWA_CONFIG_REG_v1 mma_config_corrupt = Get_XFSM_Corrupt_config(); //Receives X FSM corrupt
configuration
  mma_config_corrupt.PARITYCTRL = __MMA_PNCM_CK; //Parity checking enabled calculation disabled
  __HWA_OFFSET_REG offset_reg2 = __gen_HWA_OFFSET_REG();
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWAOPEN(mma_config_corrupt,offset_reg2,__MMA_OPEN_FSM_RESET);
 __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWAXFER(__MMA_XFER_SRC_HWA_CONFIG);
  __HWAADV();
  __HWAADV();
  __HWAADV();
  __HWARCV(0);
  __asm(" NOP");
  __asm(" NOP");
}
```

2.3.2.5 C 读取错误

当对同一 C 矩阵组有多个同时的读取访问时，会发生 C 读取错误，这会导致冲突。使用以下步骤注入此错误：

1. HWAOPEN 指令用于通过将操作配置为 MULPLUS，写入 HWA_CONFIG 和 HWA_OFFSET。
2. 并行执行 HWAOP 及 HWAXFER 操作。
3. 对 C 矩阵组进行短暂而频繁的读取及写入会触发错误。
4. 执行 HWARCV 指令以观察错误。

```
void c_read_error(void)
{
    int switch_period=0;
    int repeat_op_count = 0;
    for( ;switch_period<5;switch_period++)
    {
        __HWA_CONFIG_REG_v1 config_reg = __gen_HWA_CONFIG_REG_v1();
        config_reg.C_OPERATION0 = __MMA_C_CONFIG_MULPLUS;
        config_reg.C_OPERATION1 = __MMA_C_CONFIG_MULPLUS;
        config_reg.C_CRSWPER = switch_period; //different switching period
        config_reg.C_CWSWPER  = switch_period+10000;
        __HWA_OFFSET_REG offset_reg;
        offset_reg = __gen_HWA_OFFSET_REG(); //This generates the offset register.
        __HWAOPEN(config_reg,offset_reg,__MMA_OPEN_FSM_RESET); //This opens theHWAOPEN
        repeat_op_count = 0;
        for(;repeat_op_count<5;repeat_op_count++)
        {
        __asm(" HWAOP.S1 SA0");
        __asm(" ||HWAXFER .L1 CMAT");
        __HWAADV();
        __HWAADV();
        __HWAADV();
        __HWAADV();
        }
    }
  __HWARCV(0); //calls the error
  __asm(" NOP");
__asm(" NOP");
}
```

2.3.2.6 C 写入错误

当对同一 C 矩阵组同时进行多个写入访问时，会发生 C 写入错误，从而导致冲突。使用以下步骤注入此错误：

1. HWAOPEN 指令用于通过将操作配置为 MULPLUS，写入 HWA_CONFIG 和 HWA_OFFSET。
2. 并行执行 HWAOP 及 HWALDC 操作。
3. 对 C 矩阵组进行短暂而频繁的读取和写入会触发错误。
4. 执行 HWARCV 指令以观察错误。

```
void c_write_error(void)
{
   int switch_period=0;
    int repeat_op_count = 0;
    for( ;switch_period<2;switch_period++)
    {
        __HWA_CONFIG_REG_v1 config_reg = __gen_HWA_CONFIG_REG_v1();
        config_reg.C_OPERATION0 = __MMA_C_CONFIG_MULPLUS;
        config_reg.C_OPERATION1 = __MMA_C_CONFIG_MULPLUS;
        config_reg.C_CRSWPER = switch_period; //different switching period
        config_reg.C_CWSWPER  = switch_period+10000;
        __HWA_OFFSET_REG offset_reg;
        offset_reg = __gen_HWA_OFFSET_REG(); //This generates the offset register.
        __HWAOPEN(config_reg,offset_reg,__MMA_OPEN_FSM_RESET); //This opens theHWAOPEN
        repeat_op_count = 0;
        for(;repeat_op_count<10;repeat_op_count++)
        {
        __asm(" HWALDC .L2 VB0"); //Earlier SE0++
        __asm(" ||HWAOP.S1 SA0"); //This should run them in parallel
        __HWAADV();
        __HWAADV();
        __HWAADV();
        __HWAADV();
        }
    }
    __HWARCV(0);  //Calls the error
    __asm(" NOP");
    __asm(" NOP");
}
```

3 流程图

状态图表示 MMA 故障注入测试的顺序执行。该图概述了介绍故障、从故障中恢复和注入下一组故障的结构化方法。

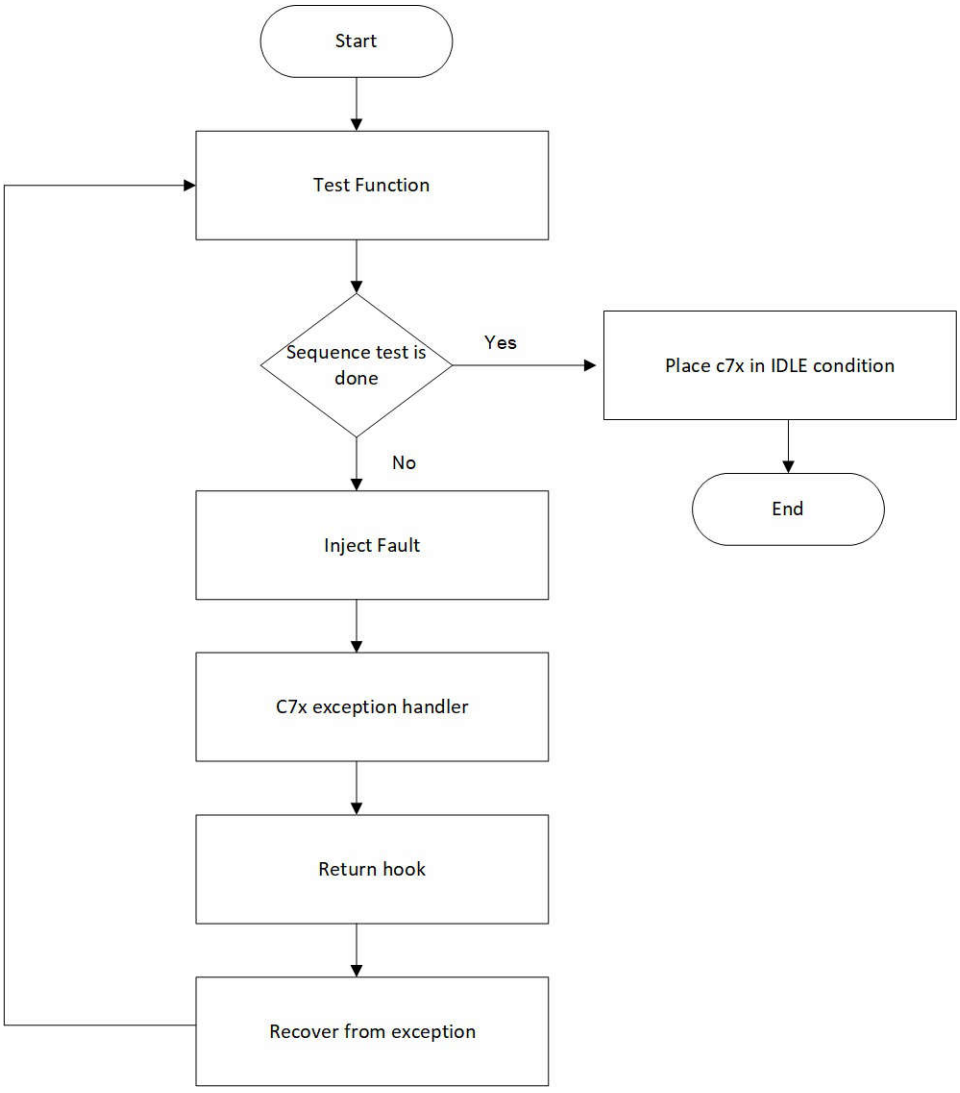

图 3-1. 测试序列流程图

3.1 代码更改

在开始实现和更改代码之前，必须遵循某些先决条件。

必须注册返回挂钩。如果未注册返回挂钩，则在异常处理程序之后不会执行返回挂钩函数。

3.1.1 返回挂钩定义

注册的返回挂钩函数必须包含以下汇编代码，以便从异常处理程序恢复，但这仅用于序列测试。对于常规流程，不得执行此操作，因为这会生成异常序列。

```
void reg_return_hook (void)
{
    __asm(" addkpc.d1 $PCR_OFFSET(test_fun), a0"); //Transfers the //test function
//address to a0.
    __asm(" ldd.d1 *ECSP[1], a1");//Status is saved in a1
    __asm(" rete.s1 a0,a1"); //rete, returns from the exception //to the beginning of the test
function, as //a0 contains the test //function address
}
```

3.1.2 清除 MMA 函数

此函数清除 MMA 内部状态，以便可以准备另一轮测试。

```
void clear_mma(void)
{
     __HWA_CONFIG_REG_v1 mma_config_reg;
     mma_config_reg = __gen_HWA_CONFIG_REG_v1();
     mma_config_reg.PARITYCTRL = __MMA_NORMAL;
     __HWA_OFFSET_REG offset_reg;
     offset_reg = __gen_HWA_OFFSET_REG();
     __HWAOPEN(mma_config_reg,offset_reg,__MMA_OPEN_FSM_RESET);
     __HWAADV();
     __HWAADV();
     __HWAADV();
     __HWAADV();
     __HWACLOSE(0); //Clearing the first error code
}
```

3.1.3 序列测试

以下函数验证是否存在 MMA 错误注入测试序列。以下代码是完全取决于用户的最终实现的示例。

```
volatile unit8_t track_exception = 0;
void test_fun(void)
{
    switch(track_exception)   /*track exception is a global variable ,        initialized to zero*/
    {
        case 0:
            clear_mma(); /*this cleares the previous mma error code, definition is given above*/
            track_exception = track_exception+1;//increment track //exception
            overflow_exception();//generate exception
            break;
        case 1:
            clear_mma();
            track_exception = track_exception+1;
            appLogPrintf("Under flow exception\n"); /*Only for debug, remove*/
            underflow_exception(); // generate exception
            break;
        case 2:
            clear_mma();
            track_exception = track_exception+1;
            appLogPrintf("offset parity exception\n"); /*only for debug remove*/
            offset_parity_test();//generate excpetion
            break;
        case 3:
            clear_mma();
            appLogPrintf("config parity exception\n"); /*Only for debug*/
            track_exception = track_exception+1;
            config_parity_test();
```

```
            break;
        case 4:
            clear_mma();
            track_exception = track_exception+1;
            appLogPrintf("offset parity exception\n");/*Only for debug*/
            offset_parity_test();
            break;
        case 5:
            appLogPrintf("Signal ESM\n");
            appLogPrintf("Waiting for ESM event\n");
            while(1) /*it needs to wait, if not the program doesnt go , and it would generate
multiple exceptions*/
            {

            }
            break;
        default:
    //Handle any unlikely scenario
            break;
}
    return;
}
```

4 总结

本文档重点说明 c7x DSP 内核上的 MMA 故障注入序列和异常处理。

本文档并未讨论在完成错误注入序列之后恢复异常以执行代码的常规流程。

5 参考资料

- 德州仪器 (TI)，*C71x DSP CPU，指令集和矩阵乘法加速器*，用户指南。
- 德州仪器 (TI)，*J721S2、TDA4AL、TDA4VL、TDA4VE、AM68A 技术参考手册*，技术参考手册。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、与某特定用途的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他安全、安保法规或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。对于因您对这些资源的使用而对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，您将全额赔偿，TI 对此概不负责。

TI 提供的产品受 TI 销售条款)、TI 通用质量指南 或 ti.com 上其他适用条款或 TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。 除非德州仪器 (TI) 明确将某产品指定为定制产品或客户特定产品，否则其产品均为按确定价格收入目录的标准通用器件。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

最后更新日期：2025 年 10 月