# **Описание используемых типов**

## **MILSTD\_type**

Указатель на структуры регистров MILSTD 0/1. Указатель на MILSTD0 имеет имя **LX\_MILSTD0**, на MILSTD1 имеет имя **LX\_MILSTD1**. Содержит следующие поля:

|  |  |
| --- | --- |
| **Имя поля** | **Описание** |
| DATA[0x400] | Поля передаваемых/принимаемых данных. Занимает 400х32 бит памяти. Однако размер передаваемых данных 16 бит, таким образом старшая часть каждого элемента массива DT не должна нести информацию. |
| CONTROL | Регистр контроля. К нему можно обращаться как к слову (.word), либо к каждому полю по отдельности (.b.<имя поля в соответствии со спецификацией> ) |
| STATUS | Регистр статуса. К нему можно обращаться как к слову (.word), либо к каждому полю по отдельности (.b.<имя поля в соответствии со спецификацией> ) |
| ERROR | Регистр ошибок. К нему можно обращаться как к слову (.word), либо к каждому полю по отдельности (.b.<имя поля в соответствии со спецификацией> ) |
| COMMWORD1 | Регистр командного слова 1(КС). К нему можно обращаться как к слову (.word), либо к каждому полю по отдельности (.b.<имя поля в соответствии со спецификацией> ) |
| COMMWORD2 | Регистр командного слова 2(КС), используется для форматов 3 и 8. К нему можно обращаться как к слову (.word), либо к каждому полю по отдельности (.b.<имя поля в соответствии со спецификацией> ) |
| MODEDATA | Регистр слова данных. Используются для форматов 5,6 и 10. |
| STATWORD1 | Регистр ответного слова 1 (ОС). К нему можно обращаться как к слову (.word), либо к каждому полю по отдельности (.b.<имя поля в соответствии со спецификацией> ) |
| STATWORD2 | Регистр ответного слова 2 (ОС), используется для форматов 3 и 8. К нему можно обращаться как к слову (.word), либо к каждому полю по отдельности (.b.<имя поля в соответствии со спецификацией> ) |
| INTEN | Регистр маскирование прерываний. К нему можно обращаться как к слову (.word), либо к каждому полю по отдельности (.b.<имя поля в соответствии со спецификацией> ) |
| MSG | Регистр декодирования сообщений |

## **MILSTD\_Init\_type**

Тип структуры для инициализации MILSTD. Содержит следующие поля:

|  |  |  |
| --- | --- | --- |
| **Имя поля** | **Описание** | **Значение** |
| Mode | Режим работы MILSTD | *MILSTD\_Mode\_BC* - КШ  *MILSTD\_Mode\_RT* - ОУ  *MILSTD\_Mode\_Monitor* – М |
| Channel | Выбор канала, по которому будет идти передача (основной или резервный) | *MILSTD\_Channel\_Main* – основной канал  *MILSTD\_Channel\_Reserv* – резервный канал |
| ResetErr | Управление сбросом ошибок в режиме ОУ и М | *MILSTD\_ResetErr\_IDLE* – автоматический сброс ошибок при установки бита IDLE регистра STATUS  *MILSTD\_ResetErr\_MR* – сброс ошибок только битом MR регистра CONTROL |
| RTA | Адрес ОУ. Содержит адрес, который присвоен устройству, если контролер работает в режиме ОУ. | От 0 до 30 |
| InvPRD | Управление сигналом EN (разрешение передачи) в физическом модуле Manchester. В зависимости от модели модуля передача разрешена либо при EN = 1, либо при EN = 0. | *MILSTD\_InvPRD\_Dis* – разрешение передачи при EN = 1  *MILSTD\_InvPRD\_En* - разрешение передачи при EN = 0 |
| Filter | Управление фильтрами. | *MILSTD\_Filter\_Dis* – фильтры выключены  *MILSTD\_Filter\_NRZ* – фильтер NRZ включен, входной фильтр помех выключен.  *MILSTD\_Filter\_Interference* – фильтр NRZ выключен, входной фильтр помех включен  *MILSTD\_Filter\_En* – оба фильтры включены |
| AutoTune | Управление автоматической подстройкой середины битовых интервалов | *MILSTD\_AutoTune\_Dis* - выключена  *MILSTD\_AutoTune\_En* - включена |

## **MILSTD\_IT\_type**

|  |  |
| --- | --- |
| **Имя** | **Значение** |
| *MILSTD\_IT\_IDLEIE* | Прерывание неактивности контроллера |
| *MILSTD\_IT\_RFLAGNIE* | Прерывание при приёме достоверного слова |
| *MILSTD\_IT\_VALMESSIE* | Прерывание при успешном завершении транзакции в канале |
| *MILSTD\_IT\_ERRIE* | Прерывание при возникновении ошибки в сообщении |
| *MILSTD\_IT\_RTAIE* | Прерывание по ошибке в адресе оконечного устройства |

# **Описание используемых функций**

## **HAL\_MILSTD\_Reset**

**Резюме**

**void** **HAL\_MILSTD\_Reset**( MILSTD\_type \*MILSTDx )

MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

**Описание**

Функция делает сброс MILSTD, путем установки бита MR в регистре CONTROL. Перед инициализацией MILSTD необходимо вызывать эту функцию.

**Пример**

#include <hal\_1967VN044.h>

int main(void)

{

HAL\_MILSTD\_Reset(LX\_MILSTD0);

}

## **HAL\_MILSTD\_Init**

**Резюме**

**void** **HAL\_MILSTD\_Init**( MILSTD\_type \*MILSTDx, MILSTD\_Init\_type \*InitStruct )

MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

MILSTD\_Init\_type \*InitStruct – указатель на структуры инициализации для MILSTD.

**Описание**

Функция инициализирует выбранный MILSTD в соответствии со структурой MILSTD\_Init\_type. Перед первой инициализацией выбранного MILSTD необходимо сначала использовать функцию HAL\_MILSTD\_Reset, а потом данную функцию. Для повторного пера инициализацией необходимо сначала использовать функцию HAL\_MILSTD\_DeInit, а потом данную функцию.

**Пример**

#include <hal\_1967VN044.h>

#define CPU\_CLK 48000

int main(void)

{

MILSTD\_Init\_type MIL0Cfg, MIL1Cfg;

//PLL Configure

HAL\_PLL\_CoreSetup(CPU\_CLK);

//GPIO MILSTD 0 Configure

HAL\_GPIO\_Init(LX\_GPIO\_PA, GPIO\_PIN\_26|GPIO\_PIN\_27|GPIO\_PIN\_28, GPIO\_PinMode\_Alt);

HAL\_GPIO\_Init(LX\_GPIO\_PB, GPIO\_PIN\_0|GPIO\_PIN\_1, GPIO\_PinMode\_Alt);

//GPIO MILSTD 1 Configure

HAL\_GPIO\_Px\_AltInit(GPIO\_PxAlt\_Data23\_16 | GPIO\_PxAlt\_Data31\_24 | GPIO\_PxAlt\_Addr7\_0 | GPIO\_PxAlt\_Addr15\_8 | GPIO\_PxAlt\_Addr21\_16, GPIO\_PxAlt\_Mode\_MILSTD);

//MILSTD 0 Init

MIL0Cfg.Mode = MILSTD\_Mode\_BC;

MIL0Cfg.Channel = MILSTD\_Channel\_Main;

MIL0Cfg.Filter = MILSTD\_Filter\_Dis;

MIL0Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL0Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

//MIL0Cfg.RTA = 1;

MIL0Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD0);

HAL\_MILSTD\_Init(LX\_MILSTD0, &MIL0Cfg);

//MILSTD 1 Init

MIL1Cfg.Mode = MILSTD\_Mode\_RT;

MIL1Cfg.Channel = MILSTD\_Channel\_Main;

MIL1Cfg.Filter = MILSTD\_Filter\_Dis;

MIL1Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL1Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

MIL1Cfg.RTA = 1;

MIL1Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD1);

HAL\_MILSTD\_Init(LX\_MILSTD1, &MIL1Cfg);

}

## **HAL\_MILSTD\_ITEnable**

**Резюме**

**void** **HAL\_MILSTD\_ITEnable**( MILSTD\_type \*MILSTDx, MILSTD\_IT\_type ITName ) MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

MILSTD\_IT\_type ITName – событие, по которому разрешается запрос к контролеру прерываний

**Описание**

Функция разрешает прерывание в соответствии с параметром ITName.

**Пример**

#include <hal\_1967VN044.h>

int main(void)

{

HAL\_MILSTD\_ITEnable(LX\_MILSTD1, MILSTD\_IT\_RFLAGNIE);

}

## **HAL\_MILSTD\_ITDisable**

**Резюме**

**void** **HAL\_MILSTD\_ITDisable**( MILSTD\_type \*MILSTDx, MILSTD\_IT\_type   
ITName )

MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

MILSTD\_IT\_type ITName – событие, по которому разрешается запрос к контролеру прерываний

**Описание**

Функция запрещает прерывание в соответствии с параметром ITName.

**Пример**

#include <hal\_1967VN044.h>

int main(void)

{

HAL\_MILSTD\_ITDisable (LX\_MILSTD1, MILSTD\_IT\_RFLAGNIE);

}

## **HAL\_MILSTD\_DeInit**

**Резюме**

**void** **HAL\_MILSTD\_Deinit**( MILSTD\_type \*MILSTDx )

MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

**Описание**

Функция делает сброс контролера MILSTD (установка бита MR) и записывает 0 регистр CONTROL.

**Пример**

#include <hal\_1967VN044.h>

int main(void)

{

HAL\_MILSTD\_DeInit(LX\_MILSTD0);

}

## **HAL\_MILSTD\_CWInit**

**Резюме**

**void** **HAL\_MILSTD\_CWInit** ( MILSTD\_type \*MILSTDx, uint32\_t NumCW, uint32\_t TxRxMode, uint32\_t RTA, uint32\_t amount, uint32\_t SubAddr )

MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

uint32\_t NumCW – номер КС (1 или 2)

uint32\_t TxRxMode – Значение бита «Прием/Передача»

uint32\_t RTA – Адрес ОУ, которому предназначено КС

uint32\_t amount – Для передачи СД - количество передаваемых слов (макс. 32, значение 0 эквивалентно передаче 32 слов). Для передачи КУ - код КУ

uint32\_t SubAddr – Содержит подадрес, по которому в памяти располагаются принимаемые или передаваемые СД. Для передачи КУ этот параметр должен быть равен 0 или 0x1F

**Описание**

Функция формирует КС. Так же КС можно формировать самостоятельно (см. разделы «Пример работы в Формате (1,2,4,5,10)»)

**Пример**

#include <hal\_1967VN044.h>

int main(void)

{

HAL\_MILSTD\_CWInit(LX\_MILSTD1, 1, 0, 1, 20, 1);

}

## **HAL\_MILSTD\_DTWrite**

**Резюме**

**void** **HAL\_MILSTD\_DTWrite**( MILSTD\_type \*MILSTDx, **void** \*Buff, uint32\_t amount, uint32\_t subaddr )

MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

**void** \*Buff – указатель на массив передаваемых данных

uint32\_t amount – количество передаваемых данных (макс. 32)

uint32\_t subaddr – номер блока в который будут записаны передаваемые данные в массиве DT. Массив DT разделен на 32 блока по 32 слова в каждом.

**Описание**

Функция записывает amount данных из указателя \*Buff, в массив DT.

**Пример**

#include <hal\_1967VN044.h>

int main(void)

{

int data\_tx[15];

HAL\_MILSTD\_DTWrite (LX\_MILSTD0, &data\_tx[0], 15, 3);

}

## **HAL\_MILSTD\_StartCWTransmit**

**Резюме**

**void** **HAL\_MILSTD\_StartCWTransmit**( MILSTD\_type \*MILSTDx )

MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

**Описание**

Функция запускает передачу КС от КШ.

**Пример**

#include <hal\_1967VN044.h>

int main(void)

{

HAL\_MILSTD\_StartCWTransmit (LX\_MILSTD1);

}

## **HAL\_MILSTD\_DRRead**

**Резюме**

**void** **HAL\_MILSTD\_DRRead**( MILSTD\_type \*MILSTDx, **void** \*Buff, uint32\_t amount, uint32\_t subaddr )

MILSTD\_type \*MILSTDx – указатель на структуры регистров MILSTD 0/1

**void** \*Buff – указатель на массив передаваемых данных

uint32\_t amount – количество передаваемых данных (макс. 32)

uint32\_t subaddr – номер блока в который будут записаны передаваемые данные в массиве DT. Массив DT разделен на 32 блока по 32 слова в каждом.

**Описание**

Функция сохраняет amount слов из памяти принимаемых данных контролера MILSTD по указателю \*Buff.

**Пример 1**

#include <hal\_1967VN044.h>

int main(void)

{

int data\_rx[15];

HAL\_MILSTD\_DRRead (LX\_MILSTD1, &data\_rx[0], 15, 1);

}

# **Пример работы**

## **Пример работы в Формате 1 (КШ передает данные в ОУ)**

**#include** <hal\_1967VN044.h>

**#include** <builtins.h>

**#include** <sysreg.h>

**#define** CPU\_CLK 48000

**#define** BUS\_CLK 30000

**#define** N 20

**#define** START\_VAL 0xAAAA

**int** BCRqst = 0, ReadEn = 0, ErrCount = 0;

**int** data\_tx[N], data\_rx[N];

**void** **Error**(){

**while**(1);

}

**\_\_attribute**((interrupt))

**void** **MIL1\_HNDLR**(**void**){

BCRqst = 1;

/\*\*\*\*\*\*\* Forming StatWord \*\*\*\*\*\*\*/

ReadEn = 1;

}

**int** **main**(**void**){

**int** i, j, tmp, \*ptr;

MILSTD\_Init\_type MIL0Cfg, MIL1Cfg;

MILSTD\_COMMWORD CommWordTemp;

MILSTD\_STATWORD StatWordTemp;

//Interrupt Clear and Global Enable

HAL\_Interrupt\_IMASKClear();

HAL\_Interrupt\_ILATClear();

HAL\_Interrupt\_GlobalEnable();

\_\_RDS;

//PLL Configure

HAL\_PLL\_CoreSetup(CPU\_CLK);

HAL\_PLL\_BusSetup(BUS\_CLK);

//Forming Data Tx

**for**(i=0;i<N;i++){

data\_tx[i] = START\_VAL + i;

}

//GPIO MILSTD 0 Configure

HAL\_GPIO\_Init(LX\_GPIO\_PA, *GPIO\_PIN\_26*|*GPIO\_PIN\_27*|*GPIO\_PIN\_28*, *GPIO\_PinMode\_Alt*);

HAL\_GPIO\_Init(LX\_GPIO\_PB, *GPIO\_PIN\_0*|*GPIO\_PIN\_1*, *GPIO\_PinMode\_Alt*);

//GPIO MILSTD 1 Configure

HAL\_GPIO\_Px\_AltInit(*GPIO\_PxAlt\_Data23\_16* | *GPIO\_PxAlt\_Data31\_24* | *GPIO\_PxAlt\_Addr7\_0* | *GPIO\_PxAlt\_Addr15\_8* | *GPIO\_PxAlt\_Addr21\_16*, *GPIO\_PxAlt\_Mode\_MILSTD*);

//MILSTD 0 Init

MIL0Cfg.Mode = *MILSTD\_Mode\_BC*;

MIL0Cfg.Channel = MILSTD\_Channel\_Main;

MIL0Cfg.Filter = MILSTD\_Filter\_Dis;

MIL0Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL0Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

//MIL0Cfg.RTA = 1;

MIL0Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD0);

HAL\_MILSTD\_Init(LX\_MILSTD0, &MIL0Cfg);

//MILSTD 1 Init

MIL1Cfg.Mode = *MILSTD\_Mode\_RT*;

MIL1Cfg.Channel = MILSTD\_Channel\_Main;

MIL1Cfg.Filter = MILSTD\_Filter\_Dis;

MIL1Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL1Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

MIL1Cfg.RTA = 1;

MIL1Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD1);

HAL\_MILSTD\_Init(LX\_MILSTD1, &MIL1Cfg);

//MILSTD 1 IT Init

HAL\_Interrupt\_Enable(intMIL1, &MIL1\_HNDLR);

HAL\_MILSTD\_ITEnable(LX\_MILSTD1, *MILSTD\_IT\_RFLAGNIE*);

//MILSTD 1 SW RTA

StatWordTemp.b.RTA = LX\_MILSTD1->CONTROL.b.RTA;

LX\_MILSTD1->STATWORD1.word = StatWordTemp.word;

// MILSTD 0 CW Init

CommWordTemp.b.RTA = 1;

CommWordTemp.b.TXRX = 0;

CommWordTemp.b.SUBADDR = 2;

CommWordTemp.b.CODECOMM = N;

LX\_MILSTD0->COMMWORD1.word = CommWordTemp.word;

// Write Data Tx to transfer buffer

HAL\_MILSTD\_DTWrite(LX\_MILSTD0, &data\_tx[0], N, CommWordTemp.b.SUBADDR);

// Start Tx

HAL\_MILSTD\_StartCWTransmit(LX\_MILSTD0);

// While COMMWORD

**while**(!BCRqst);

// Wait Message

**while**(!LX\_MILSTD1->STATUS.b.VALMESS);

// Read Data

**if** (ReadEn){

HAL\_MILSTD\_DRRead(LX\_MILSTD1, &data\_rx[0], N, LX\_MILSTD1->COMMWORD1.b.SUBADDR);

// Check Data

**for**(i=0;i<N;i++){

**if** (data\_tx[i]!=data\_rx[i]) ErrCount++;

}

}

**else** Error();

**if** (ErrCount) Error();

**return** 0;

}

В данной программе приведен пример работы передатчика и приемника в рамках одного процессора (т.е MILSTD0 передает данные в MILSTD1). Так как в действительности такая задача вряд ли возникнет, опишем действия КШ и ОУ для формата 1 по отдельности.

### **Пример работы КШ**

КШ. Для формата 1, КШ – передатчик. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_BC.*
3. Сформировать COMMWORD1, обращаю внимание на то, что регистр COMMWORD для КШ- Передатчика работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_COMMWORD CommWordTemp» и формируем ее значение, при помощи битовых полей, далее перезаписывает CommWordTemp.word в LX\_MILSTDx->COMMWORD1.word.
4. Записать передаваемые данные в один из банков поля DT(номер банка соответствует полю SUBADDR регистра COMMWORD). Размер банка 32х16, таким образом за одну транзакцию нельзя передать больше 32 слов MILSTD.
5. Запустить передачу данных.
6. Ожидать ответа. По приходу ответа проанализировать STATWORD1, в соответствии с полученным STATWORD1 выполнить необходимые действия.

### **Пример работы ОУ**

Для формата 1, ОУ – приемник. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_RT.*
3. Записать в поле RTA регистра STATWORD1, адрес своего ОУ( т.е поле RTA регистра CONTROL). Обращаю внимание на то, что регистр STATWORD для ОУ-Приемника работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_STATWORD StatWordTemp» и формируем значение поля RTA, далее перезаписывает StatWordTemp.word в LX\_MILSTDx->STATWORD1.word.
4. Ожидать запроса от КШ. Для этого в данном примере включится прерывание HAL\_MILSTD\_ITEnable(LX\_MILSTDx, MILSTD\_IT\_RFLAGNIE);
5. При приходе КС от КШ произойдет прерывание (прерывание НЕ произойдет если поля RTA в КС и ОС не будут равны или если КС не является групповым), далее будет от 4 до 12 мкс для анализ принятого КС (КС будет помещено в регистр КС данного ОУ) и формирования ОС. Спустя 4-12 мкс в КШ будет отправлено ОС.
6. Если необходимо, вычитать принятые данные (перед этим желательно убедится, что бит VALLMESS регистра STATUS установлен) из соответствующего банка памяти поля DR. Номер банка памяти в которые записаны принятые данные = COMMWORD1.b.SUBADDR.

## **Пример работы в Формате 2 (КШ принимает данные от ОУ)**

**#define** CPU\_CLK 48000

**#define** BUS\_CLK 30000

**#define** N 20

**#define** START\_VAL 0xAAAA

**int** BCRqst = 0, ReadEn = 0, ErrCount = 0, TDRqst = 0;

**int** data\_tx[N];

**int** data\_rx[N];

**void** **Error**(**int** ErrCode){

\_\_NOP;

**while**(1);

}

**\_\_attribute**((interrupt))

**void** **MIL0\_HNDLR**(**void**){

TDRqst = 1;

// Analysis STATWORD //

ReadEn = 1;

}

**\_\_attribute**((interrupt))

**void** **MIL1\_HNDLR**(**void**){

BCRqst = 1;

// Analysis COMMWORD and forming STATWORD //

}

**int** **main**(**void**){

**int** i, j, tmp, \*ptr;

MILSTD\_Init\_type MIL0Cfg, MIL1Cfg;

MILSTD\_COMMWORD CommWordTemp;

MILSTD\_STATWORD StatWordTemp;

CommWordTemp.word = 0;

StatWordTemp.word = 0;

//Interrupt Clear and Global Enable

HAL\_Interrupt\_IMASKClear();

HAL\_Interrupt\_ILATClear();

HAL\_Interrupt\_GlobalEnable();

\_\_RDS;

//Flag Enable

HAL\_SYS\_FlagEnable();

//\_\_builtins\_sysreg\_write(\_\_FLAGREG, 0xFE);

//PLL Configure

HAL\_PLL\_CoreSetup(CPU\_CLK);

HAL\_PLL\_BusSetup(BUS\_CLK);

//Forming Data Tx

**for**(i=0;i<N;i++){

data\_tx[i] = START\_VAL + i;

}

//GPIO MILSTD 0 Configure

HAL\_GPIO\_Init(LX\_GPIO\_PA, *GPIO\_PIN\_26*|*GPIO\_PIN\_27*|*GPIO\_PIN\_28*, *GPIO\_PinMode\_Alt*);

HAL\_GPIO\_Init(LX\_GPIO\_PB, *GPIO\_PIN\_0*|*GPIO\_PIN\_1*, *GPIO\_PinMode\_Alt*);

//GPIO MILSTD 1 Configure

HAL\_GPIO\_Px\_AltInit(*GPIO\_PxAlt\_Data23\_16* | *GPIO\_PxAlt\_Data31\_24* | *GPIO\_PxAlt\_Addr7\_0* | *GPIO\_PxAlt\_Addr15\_8* | *GPIO\_PxAlt\_Addr21\_16*, *GPIO\_PxAlt\_Mode\_MILSTD*);

//MILSTD 0 Init

MIL0Cfg.Mode = *MILSTD\_Mode\_BC*;

MIL0Cfg.Channel = MILSTD\_Channel\_Main;

MIL0Cfg.Filter = MILSTD\_Filter\_Dis;

MIL0Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL0Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

//MIL0Cfg.RTA = 1;

MIL0Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD0);

HAL\_MILSTD\_Init(LX\_MILSTD0, &MIL0Cfg);

//MILSTD 1 IT Init

HAL\_Interrupt\_Enable(intMIL0, &MIL0\_HNDLR);

HAL\_MILSTD\_ITEnable(LX\_MILSTD0, *MILSTD\_IT\_RFLAGNIE*);

//MILSTD 1 Init

MIL1Cfg.Mode = *MILSTD\_Mode\_RT*;

MIL1Cfg.Channel = MILSTD\_Channel\_Main;

MIL1Cfg.Filter = MILSTD\_Filter\_Dis;

MIL1Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL1Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

MIL1Cfg.RTA = 1;

MIL1Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD1);

HAL\_MILSTD\_Init(LX\_MILSTD1, &MIL1Cfg);

//MILSTD 1 IT Init

HAL\_Interrupt\_Enable(intMIL1, &MIL1\_HNDLR);

HAL\_MILSTD\_ITEnable(LX\_MILSTD1, *MILSTD\_IT\_RFLAGNIE*);

//MILSTD 1 SW RTA

StatWordTemp.b.RTA = LX\_MILSTD1->CONTROL.b.RTA;

LX\_MILSTD1->STATWORD1.word = StatWordTemp.word;

// MILSTD 0 CW Init

CommWordTemp.b.RTA = 1;

CommWordTemp.b.TXRX = 1;

CommWordTemp.b.SUBADDR = 1;

CommWordTemp.b.CODECOMM = N;

LX\_MILSTD0->COMMWORD1.word = CommWordTemp.word;

// CW Transmit

HAL\_SYS\_WaitClk(30);

HAL\_MILSTD\_StartCWTransmit(LX\_MILSTD0);

**while**(!BCRqst);

**if** (LX\_MILSTD1->COMMWORD1.b.TXRX) HAL\_MILSTD\_DTWrite(LX\_MILSTD1, &data\_tx[0], N, LX\_MILSTD1>COMMWORD1.b.SUBADDR);

**else** Error(1);

**while**(!TDRqst);

// Wait Message

**while**(!LX\_MILSTD0->STATUS.b.VALMESS);

**if** (ReadEn){

HAL\_MILSTD\_DRRead(LX\_MILSTD0, &data\_rx[0], N, CommWordTemp.b.SUBADDR);

**for**(i=0;i<N;i++){

**if** (data\_tx[i]!=data\_rx[i]) ErrCount++;

}

}

**else** Error(2);

**if** (ErrCount) Error(3);

**return** 0;

}

В данной программе приведен пример работы передатчика и приемника в рамках одного процессора (т.е MILSTD1 передает данные в MILSTD0). Так как в действительности такая задача вряд ли возникнет, опишем действия КШ и ОУ для формата 2 по отдельности.

### **Пример работы КШ**

КШ. Для формата 2, КШ – приемник. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_BC.*
3. Сформировать COMMWORD1, обращаю внимание на то, что регистр COMMWORD для КШ- Передатчика работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_COMMWORD CommWordTemp» и формируем ее значение, при помощи битовых полей, далее перезаписывает CommWordTemp.word в LX\_MILSTDx->COMMWORD1.word. Для того, чтобы КШ был приемником в COMMWORD1 необходимо установить бит TXRX.
4. Подождать некоторое время (в данном примере ожидание более 30 тактов), после запустить транзакцию с помощью функции HAL\_MILSTD\_StartCWTransmit(LX\_MILSTDx)
5. Ожидать ОС. Для это можно установить прерывание по событие RFLAGN.
6. По приходу ОС, проанализировать его и, если необходимо, вычитать данные из буфера приемника (предварительно убедившись, что бит VALMESS регистра STATUS установлен).

### **Пример работы ОУ**

Для формата 2, ОУ – передатчик. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_RT.*
3. Записать в поле RTA регистра STATWORD1, адрес своего ОУ( т.е поле RTA регистра CONTROL). Обращаю внимание на то, что регистр STATWORD для ОУ-Приемника работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_STATWORD StatWordTemp» и формируем значение поля RTA, далее перезаписывает StatWordTemp.word в LX\_MILSTDx->STATWORD1.word.
4. Ожидать запроса от КШ. Для этого можно включить прерывание HAL\_MILSTD\_ITEnable(LX\_MILSTD1, MILSTD\_IT\_RFLAGNIE);
5. При приходе КС от КШ произойдет прерывание (прерывание НЕ произойдет если поля RTA в КС и ОС не будут равны или если КС не является групповым), далее будет от 4 до 12 мкс для анализ принятого КС (КС будет помещено в регистр КС данного ОУ) и формирования ОС и записи данных в буфер передатчика (данные так же можно положить в буфер передатчика и до прихода запроса от КШ). Спустя 4-12 мкс в КШ будет отправлено ОС и данные.

## **Пример работы в Формате 4 (КШ передает КС, которое содержит КУ, ОУ в ответ посылает ОС)**

**#define** CPU\_CLK 48000

**#define** BUS\_CLK 30000

**int** BCRqst = 0, ReadEn = 0, ErrCount = 0, TDRqst = 0;

**void** **Error**(**int** ErrCode){

\_\_NOP;

**while**(1);

}

**\_\_attribute**((interrupt))

**void** **MIL0\_HNDLR**(**void**){

TDRqst = 1;

}

**\_\_attribute**((interrupt))

**void** **MIL1\_HNDLR**(**void**){

BCRqst = 1;

}

**int** **main**(**void**){

**int** i, j, tmp, \*ptr;

MILSTD\_Init\_type MIL0Cfg, MIL1Cfg;

MILSTD\_COMMWORD CommWordTemp;

MILSTD\_STATWORD StatWordTemp;

CommWordTemp.word = 0;

StatWordTemp.word = 0;

//Interrupt Clear and Global Enable

HAL\_Interrupt\_IMASKClear();

HAL\_Interrupt\_ILATClear();

HAL\_Interrupt\_GlobalEnable();

\_\_RDS;

//Flag Enable

HAL\_SYS\_FlagEnable();

//\_\_builtins\_sysreg\_write(\_\_FLAGREG, 0xFE);

//PLL Configure

HAL\_PLL\_CoreSetup(CPU\_CLK);

HAL\_PLL\_BusSetup(BUS\_CLK);

//GPIO MILSTD 0 Configure

HAL\_GPIO\_Init(LX\_GPIO\_PA, *GPIO\_PIN\_26*|*GPIO\_PIN\_27*|*GPIO\_PIN\_28*, *GPIO\_PinMode\_Alt*);

HAL\_GPIO\_Init(LX\_GPIO\_PB, *GPIO\_PIN\_0*|*GPIO\_PIN\_1*, *GPIO\_PinMode\_Alt*);

//GPIO MILSTD 1 Configure

HAL\_GPIO\_Px\_AltInit(*GPIO\_PxAlt\_Data23\_16* | *GPIO\_PxAlt\_Data31\_24* | *GPIO\_PxAlt\_Addr7\_0* | *GPIO\_PxAlt\_Addr15\_8* | *GPIO\_PxAlt\_Addr21\_16*, *GPIO\_PxAlt\_Mode\_MILSTD*);

//MILSTD 0 Init

MIL0Cfg.Mode = *MILSTD\_Mode\_BC*;

MIL0Cfg.Channel = MILSTD\_Channel\_Main;

MIL0Cfg.Filter = MILSTD\_Filter\_Dis;

MIL0Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL0Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

//MIL0Cfg.RTA = 1;

MIL0Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD0);

HAL\_MILSTD\_Init(LX\_MILSTD0, &MIL0Cfg);

//MILSTD 1 IT Init

HAL\_Interrupt\_Enable(intMIL0, &MIL0\_HNDLR);

HAL\_MILSTD\_ITEnable(LX\_MILSTD0, *MILSTD\_IT\_RFLAGNIE*);

//MILSTD 1 Init

MIL1Cfg.Mode = *MILSTD\_Mode\_RT*;

MIL1Cfg.Channel = MILSTD\_Channel\_Main;

MIL1Cfg.Filter = MILSTD\_Filter\_Dis;

MIL1Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL1Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

MIL1Cfg.RTA = 1;

MIL1Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD1);

HAL\_MILSTD\_Init(LX\_MILSTD1, &MIL1Cfg);

//MILSTD 1 IT Init

HAL\_Interrupt\_Enable(intMIL1, &MIL1\_HNDLR);

HAL\_MILSTD\_ITEnable(LX\_MILSTD1, *MILSTD\_IT\_RFLAGNIE*);

//MILSTD 1 SW RTA

StatWordTemp.b.RTA = LX\_MILSTD1->CONTROL.b.RTA;

LX\_MILSTD1->STATWORD1.word = StatWordTemp.word;

// MILSTD 0 CW Init

CommWordTemp.b.RTA = 1;

CommWordTemp.b.TXRX = 1;

CommWordTemp.b.SUBADDR = 0;//0x1F;

CommWordTemp.b.CODECOMM = 2;

LX\_MILSTD0->COMMWORD1.word = CommWordTemp.word;

// CW Transmit

HAL\_SYS\_WaitClk(30);

HAL\_MILSTD\_StartCWTransmit(LX\_MILSTD0);

**while**(!BCRqst);

// Forming STATWORD

**if** (LX\_MILSTD1->COMMWORD1.b.CODECOMM == 2){

StatWordTemp.b.TRAN\_ANSWER\_WORD = 1;

LX\_MILSTD1->STATWORD1.word = StatWordTemp.word;

}

**else** Error(1);

**while**(!TDRqst);

// Wait Message

**while**(!LX\_MILSTD0->STATUS.b.VALMESS);

// Check AW

**if** (!LX\_MILSTD0->STATWORD1.b.TRAN\_ANSWER\_WORD) Error(2);

**while**(1);

**return** 0;

}

В данной программе приведен пример работы КШ и ОУ в рамках одного процессора (т.е MILSTD0 – КШ, MILSTD1 – ОУ). Так как в действительности такая задача вряд ли возникнет, опишем действия КШ и ОУ для формата 4 по отдельности.

### **Пример работы КШ**

КШ. Для формата 4, КШ формирует КС содержащее КУ. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_BC.*
3. Сформировать COMMWORD1, обращаю внимание на то, что регистр COMMWORD для КШ- Передатчика работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_COMMWORD CommWordTemp» и формируем ее значение, при помощи битовых полей, далее перезаписывает CommWordTemp.word в LX\_MILSTDx->COMMWORD1.word. Для того, чтобы работать в Формате 4 в COMMWORD1 необходимо установить бит TXRX.
4. Подождать некоторое время (в данном примере ожидание более 30 тактов), после запустить транзакцию с помощью функции HAL\_MILSTD\_StartCWTransmit(LX\_MILSTDx)
5. Ожидать ОС. Для это можно установить прерывание по событие RFLAGN.
6. По приходу ОС, проанализировать его.

### **Пример работы ОУ**

Для формата 4, ОУ –анализирует КУ и отправляет ОС. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_RT.*
3. Записать в поле RTA регистра STATWORD1, адрес своего ОУ( т.е поле RTA регистра CONTROL). Обращаю внимание на то, что регистр STATWORD для ОУ-Приемника работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_STATWORD StatWordTemp» и формируем значение поля RTA, далее перезаписывает StatWordTemp.word в LX\_MILSTDx->STATWORD1.word.
4. Ожидать запроса от КШ. Для этого можно включить прерывание HAL\_MILSTD\_ITEnable(LX\_MILSTDx, MILSTD\_IT\_RFLAGNIE);
5. При приходе КС от КШ произойдет прерывание (прерывание НЕ произойдет если поля RTA в КС и ОС не будут равны или если КС не является групповым), далее будет от 4 до 12 мкс для анализ принятого КС содержащего КУ (КС будет помещено в регистр КС данного ОУ, в регистр MSG будет помещен код сообщения соответствующий описание из спецификации, Таблица 217 – Коды сообщений регистра MSG) и формирования ОС (в данной программе для примера выставляется бит «Передача ответного слова»). Спустя 4-12 мкс в КШ будет отправлено ОС.

## **Пример работы в Формате 5 (КШ передает КС, которое содержит КУ, ОУ в ответ посылает ОС и СД)**

**#define** CPU\_CLK 48000

**#define** BUS\_CLK 30000

**#define** WORDDATA 0xF31E

**int** BCRqst = 0, ReadEn = 0, ErrCount = 0, TDRqst = 0;

**int** WordData;

**void** **Error**(**int** ErrCode){

\_\_NOP;

**while**(1);

}

**\_\_attribute**((interrupt))

**void** **MIL0\_HNDLR**(**void**){

TDRqst = 1;

}

**\_\_attribute**((interrupt))

**void** **MIL1\_HNDLR**(**void**){

BCRqst = 1;

}

**int** **main**(**void**){

**int** i, j, tmp, \*ptr;

MILSTD\_Init\_type MIL0Cfg, MIL1Cfg;

MILSTD\_COMMWORD CommWordTemp;

MILSTD\_STATWORD StatWordTemp;

CommWordTemp.word = 0;

StatWordTemp.word = 0;

//Interrupt Clear and Global Enable

HAL\_Interrupt\_IMASKClear();

HAL\_Interrupt\_ILATClear();

HAL\_Interrupt\_GlobalEnable();

\_\_RDS;

//Flag Enable

HAL\_SYS\_FlagEnable();

//\_\_builtins\_sysreg\_write(\_\_FLAGREG, 0xFE);

//PLL Configure

HAL\_PLL\_CoreSetup(CPU\_CLK);

HAL\_PLL\_BusSetup(BUS\_CLK);

//GPIO MILSTD 0 Configure

HAL\_GPIO\_Init(LX\_GPIO\_PA, *GPIO\_PIN\_26*|*GPIO\_PIN\_27*|*GPIO\_PIN\_28*, *GPIO\_PinMode\_Alt*);

HAL\_GPIO\_Init(LX\_GPIO\_PB, *GPIO\_PIN\_0*|*GPIO\_PIN\_1*, *GPIO\_PinMode\_Alt*);

//GPIO MILSTD 1 Configure

HAL\_GPIO\_Px\_AltInit(*GPIO\_PxAlt\_Data23\_16* | *GPIO\_PxAlt\_Data31\_24* | *GPIO\_PxAlt\_Addr7\_0* | *GPIO\_PxAlt\_Addr15\_8* | *GPIO\_PxAlt\_Addr21\_16*, *GPIO\_PxAlt\_Mode\_MILSTD*);

//MILSTD 0 Init

MIL0Cfg.Mode = *MILSTD\_Mode\_BC*;

MIL0Cfg.Channel = MILSTD\_Channel\_Main;

MIL0Cfg.Filter = MILSTD\_Filter\_Dis;

MIL0Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL0Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

//MIL0Cfg.RTA = 1;

MIL0Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD0);

HAL\_MILSTD\_Init(LX\_MILSTD0, &MIL0Cfg);

//MILSTD 1 IT Init

HAL\_Interrupt\_Enable(intMIL0, &MIL0\_HNDLR);

HAL\_MILSTD\_ITEnable(LX\_MILSTD0, *MILSTD\_IT\_RFLAGNIE*);

//MILSTD 1 Init

MIL1Cfg.Mode = *MILSTD\_Mode\_RT*;

MIL1Cfg.Channel = MILSTD\_Channel\_Main;

MIL1Cfg.Filter = MILSTD\_Filter\_Dis;

MIL1Cfg.InvPRD = MILSTD\_InvPRD\_Dis;

MIL1Cfg.AutoTune = MILSTD\_AutoTune\_Dis;

MIL1Cfg.RTA = 1;

MIL1Cfg.ResetErr = MILSTD\_ResetErr\_IDLE;

HAL\_MILSTD\_Reset(LX\_MILSTD1);

HAL\_MILSTD\_Init(LX\_MILSTD1, &MIL1Cfg);

//MILSTD 1 IT Init

HAL\_Interrupt\_Enable(intMIL1, &MIL1\_HNDLR);

HAL\_MILSTD\_ITEnable(LX\_MILSTD1, *MILSTD\_IT\_RFLAGNIE*);

//MILSTD 1 SW RTA

StatWordTemp.b.RTA = LX\_MILSTD1->CONTROL.b.RTA;

LX\_MILSTD1->STATWORD1.word = StatWordTemp.word;

// MILSTD 0 CW Init

CommWordTemp.b.RTA = 1;

CommWordTemp.b.TXRX = 1;

CommWordTemp.b.SUBADDR = 0;//0x1F;

CommWordTemp.b.CODECOMM = 16;

LX\_MILSTD0->COMMWORD1.word = CommWordTemp.word;

// CW Transmit

HAL\_SYS\_WaitClk(30);

HAL\_MILSTD\_StartCWTransmit(LX\_MILSTD0);

**while**(!BCRqst);

// Forming STATWORD

**if** (LX\_MILSTD1->COMMWORD1.b.CODECOMM == 16){

StatWordTemp.b.TRAN\_ANSWER\_WORD = 1;

LX\_MILSTD1->STATWORD1.word = StatWordTemp.word;

LX\_MILSTD1->MODEDATA = WORDDATA;

}

**else** Error(1);

**while**(!TDRqst);

// Wait Message

**while**(!LX\_MILSTD0->STATUS.b.VALMESS);

// Check AW

**if** (!LX\_MILSTD0->STATWORD1.b.TRAN\_ANSWER\_WORD) Error(2);

// Check WD

WordData = LX\_MILSTD0->MODEDATA;

**if** (WordData != WORDDATA) Error(3);

**while**(1);

**return** 0;

}

В данной программе приведен пример работы КШ и ОУ в рамках одного процессора (т.е MILSTD0 – КШ, MILSTD1 – ОУ). Так как в действительности такая задача вряд ли возникнет, опишем действия КШ и ОУ для формата 5 по отдельности.

### **Пример работы КШ**

КШ. Для формата 5, КШ формирует КС содержащее КУ. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_BC.*
3. Сформировать COMMWORD1, обращаю внимание на то, что регистр COMMWORD для КШ- Передатчика работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_COMMWORD CommWordTemp» и формируем ее значение, при помощи битовых полей, далее перезаписывает CommWordTemp.word в LX\_MILSTDx->COMMWORD1.word. Для того, чтобы работать в Формате 5 в COMMWORD1 необходимо установить бит TXRX.
4. Подождать некоторое время (в данном примере ожидание более 30 тактов), после запустить транзакцию с помощью функции HAL\_MILSTD\_StartCWTransmit(LX\_MILSTDx)
5. Ожидать ОС. Для это можно установить прерывание по событие RFLAGN.
6. По приходу ОС, проанализировать его, и если необходимо прочитать СД из регистра MODEDATA

### **Пример работы ОУ**

Для формата 5, ОУ –анализирует КУ, отправляет ОС и СД. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_RT.*
3. Записать в поле RTA регистра STATWORD1, адрес своего ОУ( т.е поле RTA регистра CONTROL). Обращаю внимание на то, что регистр STATWORD для ОУ-Приемника работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_STATWORD StatWordTemp» и формируем значение поля RTA, далее перезаписывает StatWordTemp.word в LX\_MILSTDx->STATWORD1.word.
4. Ожидать запроса от КШ. Для этого можно включить прерывание HAL\_MILSTD\_ITEnable(LX\_MILSTDx, MILSTD\_IT\_RFLAGNIE);
5. При приходе КС от КШ произойдет прерывание (прерывание НЕ произойдет если поля RTA в КС и ОС не будут равны или если КС не является групповым), далее будет от 4 до 12 мкс для анализ принятого КС содержащего КУ (КС будет помещено в регистр КС данного ОУ, в регистр MSG будет помещен код сообщения соответствующий описание из спецификации, Таблица 217 – Коды сообщений регистра MSG), формирования ОС (в данной программе для примера выставляется бит «Передача ответного слова») и записи СД в регистр MODEDATA. Спустя 4-12 мкс в КШ будет отправлено ОС и СД.

## **Пример работы в Формате 10(КШ передает групповое КС, которое содержит КУ, и передает групповое СД)**

**#define** CPU\_CLK 48000

**#define** BUS\_CLK 30000

**#define** WORDDATA 0x01

**#define** COMM\_CTRL 0x14

**int** BCRqst = 0, ReadEn = 0, ErrCount = 0;

**int** WordData;

**void** **Error**(**int** ErrCode){

\_\_NOP;

**while**(1);

}

**\_\_attribute**((interrupt))

**void** **MIL1\_HNDLR**(**void**){

BCRqst = 1;

}

**int** **main**(**void**){

**int** i, j, tmp, \*ptr;

MILSTD\_Init\_type MIL0Cfg, MIL1Cfg;

MILSTD\_COMMWORD CommWordTemp;

MILSTD\_STATWORD StatWordTemp;

CommWordTemp.word = 0;

StatWordTemp.word = 0;

//Interrupt Clear and Global Enable

HAL\_Interrupt\_IMASKClear();

HAL\_Interrupt\_ILATClear();

HAL\_Interrupt\_GlobalEnable();

\_\_RDS;

//Flag Enable

HAL\_SYS\_FlagEnable();

//\_\_builtins\_sysreg\_write(\_\_FLAGREG, 0xFE);

//PLL Configure

HAL\_PLL\_CoreSetup(CPU\_CLK);

HAL\_PLL\_BusSetup(BUS\_CLK);

//GPIO MILSTD 0 Configure

HAL\_GPIO\_Init(LX\_GPIO\_PA, *GPIO\_PIN\_26*|*GPIO\_PIN\_27*|*GPIO\_PIN\_28*, *GPIO\_PinMode\_Alt*);

HAL\_GPIO\_Init(LX\_GPIO\_PB, *GPIO\_PIN\_0*|*GPIO\_PIN\_1*, *GPIO\_PinMode\_Alt*);

//GPIO MILSTD 1 Configure

HAL\_GPIO\_Px\_AltInit(*GPIO\_PxAlt\_Data23\_16* | *GPIO\_PxAlt\_Data31\_24* | *GPIO\_PxAlt\_Addr7\_0* | *GPIO\_PxAlt\_Addr15\_8* | *GPIO\_PxAlt\_Addr21\_16*, *GPIO\_PxAlt\_Mode\_MILSTD*);

//MILSTD 0 Init

MIL0Cfg.Mode = *MILSTD\_Mode\_BC*;

MIL0Cfg.Channel = *MILSTD\_Channel\_Main*;

MIL0Cfg.Filter = *MILSTD\_Filter\_Dis*;

MIL0Cfg.InvPRD = *MILSTD\_InvPRD\_Dis*;

MIL0Cfg.AutoTune = *MILSTD\_AutoTune\_Dis*;

//MIL0Cfg.RTA = 1;

MIL0Cfg.ResetErr = *MILSTD\_ResetErr\_IDLE*;

HAL\_MILSTD\_Reset(LX\_MILSTD0);

HAL\_MILSTD\_Init(LX\_MILSTD0, &MIL0Cfg);

//MILSTD 1 Init

MIL1Cfg.Mode = *MILSTD\_Mode\_RT*;

MIL1Cfg.Channel = *MILSTD\_Channel\_Main*;

MIL1Cfg.Filter = *MILSTD\_Filter\_Dis*;

MIL1Cfg.InvPRD = *MILSTD\_InvPRD\_Dis*;

MIL1Cfg.AutoTune = *MILSTD\_AutoTune\_Dis*;

MIL1Cfg.RTA = 1;

MIL1Cfg.ResetErr = *MILSTD\_ResetErr\_IDLE*;

HAL\_MILSTD\_Reset(LX\_MILSTD1);

HAL\_MILSTD\_Init(LX\_MILSTD1, &MIL1Cfg);

//MILSTD 1 IT Init

HAL\_Interrupt\_Enable(intMIL1, &MIL1\_HNDLR);

HAL\_MILSTD\_ITEnable(LX\_MILSTD1, *MILSTD\_IT\_RFLAGNIE*);

//MILSTD 1 SW RTA

StatWordTemp.b.RTA = LX\_MILSTD1->CONTROL.b.RTA;

LX\_MILSTD1->STATWORD1.word = StatWordTemp.word;

// MILSTD 0 CW Init

CommWordTemp.b.RTA = MILSTD\_RTA\_GROUP\_COMM;

CommWordTemp.b.TXRX = 0;

CommWordTemp.b.SUBADDR = 0x1F;

CommWordTemp.b.CODECOMM = COMM\_CTRL;

LX\_MILSTD0->COMMWORD1.word = CommWordTemp.word;

LX\_MILSTD0->MODEDATA = WORDDATA;

// CW Transmit

HAL\_SYS\_WaitClk(30);

HAL\_MILSTD\_StartCWTransmit(LX\_MILSTD0);

**while**(!BCRqst);

// Wait WordData

**while**(!LX\_MILSTD1->STATUS.b.VALMESS);

// Check WD

WordData = LX\_MILSTD1->MODEDATA;

**if** (WordData != WORDDATA) Error(1);

**while**(1);

**return** 0;

}

В данной программе приведен пример работы КШ и ОУ в рамках одного процессора (т.е MILSTD0 – КШ, MILSTD1 – ОУ). Так как в действительности такая задача вряд ли возникнет, опишем действия КШ и ОУ для формата 10 по отдельности.

### **Пример работы КШ**

КШ. Для формата 10, КШ формирует групповое КС, содержащее КУ, и формирует СД. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_BC.*
3. Сформировать COMMWORD1, обращаю внимание на то, что регистр COMMWORD для КШ- Передатчика работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_COMMWORD CommWordTemp» и формируем ее значение, при помощи битовых полей, далее перезаписывает CommWordTemp.word в LX\_MILSTDx->COMMWORD1.word. Для того, чтобы работать в Формате 10 в COMMWORD1 необходимо сбросить бит TXRX. Для группового формата сообщения необходимо установить все биты поля Адрес ОУ регистра COMMWORD1. В поле «Число СД / Код команды» регистра COMMWORD1, необходимо записать код команды (в примере взят код команды «Блокировать i-й передатчик»).
4. Записать СД в регистр MODEDATA.
5. Подождать некоторое время (в данном примере ожидание более 30 тактов), после запустить транзакцию с помощью функции HAL\_MILSTD\_StartCWTransmit(LX\_MILSTDx)

### **Пример работы ОУ**

Для формата 10, ОУ – анализируют КУ, и принимают СД. Таким образом необходимо:

1. Настроить GPIO для работы выбранного MILSTD
2. Проинициализировать выбранный MILSTD, выбрав в структуре инициализации (MILSTD\_Init\_type) для поля Mode значение *MILSTD\_Mode\_RT.*
3. Записать в поле RTA регистра STATWORD1, адрес своего ОУ( т.е поле RTA регистра CONTROL). Обращаю внимание на то, что регистр STATWORD для ОУ-Приемника работает ТОЛЬКО на запись, при чтении из этого регистр, будет читать ноль. Поэтому для удобства заводим переменную «MILSTD\_STATWORD StatWordTemp» и формируем значение поля RTA, далее перезаписывает StatWordTemp.word в LX\_MILSTDx->STATWORD1.word.
4. Ожидать запроса от КШ. Для этого можно включить прерывание HAL\_MILSTD\_ITEnable(LX\_MILSTDx, MILSTD\_IT\_RFLAGNIE);
5. При приходе КС от КШ произойдет прерывание (прерывание НЕ произойдет если поля RTA в КС и ОС не будут равны или если КС не является групповым). Проанализировать КУ, выполнить соответствующие действия. Прочитать СД.