(Register0)

BIT LOCATION	BIT ID	NAME	DEFINITION
31	INT	Int-N or Frac-N Mode Control	0 = Enables the fractional-N mode 1 = Enables the integer-N mode The LDF bit must also be set to the appropriate mode.

Tamsayı-N Modu (Integer-N Mode): Tamsayı-N modunda, PLL'nin çıkış frekansı, referans frekansının tam katları olarak ayarlanır. Bu mod, aşağıdaki formülle ifade edilir:

$$f_{OUT} = N \times f_{REF}$$

Tamsayı-N modu, genellikle daha düşük faz gürültüsü ve daha iyi spuriyus performansı sunar çünkü çıkış frekansı, referans frekansının tam katları olarak üretilir ve bu da faz kilidinin daha stabil olmasını sağlar.

Set DIVA value property based on fRFOUTA and DIVA register table (register 4[22.20])

(Register4)

			1 = Fundamental
22:20	DIVA[2:0]	RFOUT_ Output Divider Mode	Sets RFOUT_ output divider mode. Double buffered by register 0 when REG4DB = 1. 000 = Divide by 1, if 3000MHz ≤ fRFOUTA ≤ 6000MHz 001 = Divide by 2, if 1500MHz ≤ fRFOUTA < 3000MHz 010 = Divide by 4, if 750MHz ≤ fRFOUTA < 1500MHz 011 = Divide by 8, if 375MHz ≤ fRFOUTA < 750MHz 100 = Divide by 16, if 187.5MHz ≤ fRFOUTA < 375MHz 101 = Divide by 32, if 93.75MHz ≤ fRFOUTA < 187.5MHz 110 = Divide by 64, if 46.875MHz ≤ fRFOUTA < 93.75MHz 111 = Divide by 128, if 23.5MHz ≤ fRFOUTA < 46.875MHz

Kesirli-N Modu (Fractional-N Mode): Kesirli-N modunda, PLL'nin çıkış frekansı, referans frekansının kesirli katları olarak ayarlanabilir. Bu, PLL'nin çok daha ince frekans adımlarıyla ayarlanmasına olanak tanır ve aşağıdaki formülle ifade edilir:

$$f_{OUT} = (N + \frac{F}{M}) \times f_{REF}$$

(Register0)

30:15	N[15:0]	Integer Division Value	Sets integer part (N-divider) of the feedback divider factor. All integer values from 16 to 65,535 are allowed for integer mode. Integer values from 19 to 4,091 are allowed for fractional mode.

(Register1)

14:3	M[11:0]	Modulus Value (M)	Fractional modulus value used to program fVCO. See the *Int, Frac, Mod, and R Counter Relationship* section. Double buffered by register 0. 000000000000 = Unused 000000000001 = Unused 000000000010 = 2 ----- 111111111111 = 4095

Kesirli-N modu, daha geniş bir frekans aralığında ve daha küçük frekans adımlarında sentez yapabilme yeteneği sunar. Bu mod, özellikle geniş bantlı uygulamalar ve frekans atlamalı iletişim sistemleri gibi uygulamalar için idealdir. Ancak, kesirli bölme işlemi nedeniyle faz gürültüsü ve spuriyus seviyeleri tamsayı-N moduna göre biraz daha yüksek olabilir.

```
void PrintErrorCode(byte value) {
  switch (value) {
    case MAX2870_ERROR_NONE:
      break;
    case MAX2870_ERROR_STEP_FREQUENCY_EXCEEDS_PFD:
      Serial.println(F("Step frequency exceeds PFD frequency"));
      break;
    case MAX2870_ERROR_RF_FREQUENCY:
      Serial.println(F("RF frequency out of range"));
      break;
    case MAX2870_ERROR_POWER_LEVEL:
      Serial.println(F("Power level incorrect"));
      break;
    case MAX2870_ERROR_AUX_POWER_LEVEL:
      Serial.println(F("Auxiliary power level incorrect"));
      break;
    case MAX2870_ERROR_AUX_FREQ_DIVIDER:
      Serial.println(F("Auxiliary frequency divider incorrect"));
      break;
    case MAX2870_ERROR_ZERO_PFD_FREQUENCY:
      Serial.println(F("PFD frequency is zero"));
      break;
    case MAX2870_ERROR_MOD_RANGE:
      Serial.println(F("Mod is out of range"));
      break;
    case MAX2870_ERROR_FRAC_RANGE:
      Serial.println(F("Fraction is out of range"));
      break;
    case MAX2870_ERROR_N_RANGE:
```

```
      Serial.println(F("N is of range"));
      break;
    case MAX2870_ERROR_N_RANGE_FRAC:
      Serial.println(F("N is out of range under fractional mode"));
      break;
    case MAX2870_ERROR_RF_FREQUENCY_AND_STEP_FREQUENCY_HAS_REMAINDER:
      Serial.println(F("RF frequency and step frequency division has
remainder"));
      break;
    case MAX2870_ERROR_PFD_EXCEEDED_WITH_FRACTIONAL_MODE:
      Serial.println(F("PFD exceeds 50 MHz under fractional mode"));
      break;
    case MAX2870_ERROR_PRECISION_FREQUENCY_CALCULATION_TIMEOUT:
      Serial.println(F("Precision frequency calculation timeout"));
      break;
    case MAX2870_WARNING_FREQUENCY_ERROR:
      Serial.println(F("Actual frequency is different than desired"));
      break;
    case MAX2870_ERROR_DOUBLER_EXCEEDED:
      Serial.println(F("Reference frequency with doubler exceeded"));
      break;
    case MAX2870_ERROR_R_RANGE:
      Serial.println(F("R divider is out of range"));
      break;
    case MAX2870_ERROR_REF_FREQUENCY:
      Serial.println(F("Reference frequency is out of range"));
      break;
    case MAX2870_ERROR_REF_MULTIPLIER_TYPE:
      Serial.println(F("Reference multiplier type is incorrect"));
      break;
    case MAX2870_ERROR_PFD_AND_STEP_FREQUENCY_HAS_REMAINDER:
      Serial.println(F("PFD and step frequency division has remainder"));
      break;
    case MAX2870_ERROR_PFD_LIMITS:
      Serial.println(F("PFD frequency is out of range"));
      break;
  }
}
```

Birkaç error-case örneği

```
case MAX2870_ERROR_POWER_LEVEL:
      Serial.println(F("Power level incorrect"));
      break;
```

```
590  int MAX2870::setPowerLevel(uint8_t PowerLevel) {
591    if (PowerLevel < 0 && PowerLevel > 4) return MAX2870_ERROR_POWER_LEVEL;
592    if (PowerLevel == 0) {
593      MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(5, 1, MAX2870_R[0x04], 0);
594    }
595    else {
596      PowerLevel--;
597      MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(5, 1, MAX2870_R[0x04], 1);
598      MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(3, 2, MAX2870_R[0x04], PowerLevel);
599    }
600    WriteRegs();
601    return MAX2870_ERROR_NONE;
602  }
```

```
case MAX2870_ERROR_REF_FREQUENCY:
      Serial.println(F("Reference frequency is out of range"));
      break;
```

```
509  int MAX2870::setrf(uint32_t f, uint16_t r, uint8_t ReferenceDivisionType)
510  {
511    if (f > 30000000UL && ReferenceDivisionType == MAX2870_REF_DOUBLE) return MAX2870_ERROR_DOUBLER_EXCEEDED;
512    if (r > 1023 || r < 1) return MAX2870_ERROR_R_RANGE;
513    if (f < MAX2870_REFIN_MIN || f > MAX2870_REFIN_MAX) return MAX2870_ERROR_REF_FREQUENCY;
514    if (ReferenceDivisionType != MAX2870_REF_UNDIVIDED && ReferenceDivisionType != MAX2870_REF_HALF && Reference
```

```
286         else if (strcmp(field, "FREQ_DIRECT") == 0) {
287           getField(field, 1);
288           word R_divider = atoi(field);
289           getField(field, 2);
290           word INT_value = atol(field);
291           getField(field, 3);
292           word MOD_value = atoi(field);
293           getField(field, 4);
294           word FRAC_value = atoi(field);
295           getField(field, 5);
296           word RF_DIVIDER_value = atoi(field);
297           getField(field, 6);
298           if (strcmp(field, "TRUE") == 0 || strcmp(field, "FALSE") == 0) {
299             bool FRACTIONAL_MODE = false;
300             if (strcmp(field, "TRUE") == 0) {
301               FRACTIONAL_MODE = true;
302             }
303             vfo.setfDirect(R_divider, INT_value, MOD_value, FRAC_value, RF_DIVIDER_value, FRACTIONAL_MODE);
304           }
305           else {
306             ValidField = false;
307           }
```

setfDirect(…..) fonksiyonu ile fonksiyona istediğimiz frekans için gerekli kombinasyonu giriyoruz ve aşağıdaki formül kullanılarak frekans değeri üretilmiş oluyor.

fOUT=(MODN×INT+FRAC)×fREF+MODR×fPFD

Bu formüldeki terimlerin anlamları ve ilgili registerlar şunlardır:

- (f_{OUT}): Çıkış frekansı.
- (N): Tamsayı bölme oranı (INT registerı tarafından belirlenir).
- (INT): Tamsayı bölme değeri (Integer division value), PLL'nin çıkış frekansını referans frekansına göre tamsayı katlarına böler.
- (FRAC): Kesirli bölme değeri (Fractional division value), PLL'nin çıkış frekansını daha küçük adımlarla ayarlamak için kullanılır.
- (MOD): Modülatör değeri (Modulus value), kesirli bölme işleminde kullanılan ve FRAC'ın maksimum değerini belirler.
- (f_{REF}): Referans frekansı.
- (R): R sayacı değeri (R Counter value), referans frekansının PFD'ye girmeden önce bölünme oranını belirler.
- (f_{PFD}): Faz frekans dedektörü frekansı (Phase frequency detector frequency).

Max2870.cpp içinde setfDirect(…)fonksiyonu:

```
void MAX2870::setfDirect(uint16_t R_divider, uint16_t INT_value, uint16_t MOD_value, uint16_t FRAC_value, uint8_t RF_DIVIDER_value, bool FRACTIONAL_MODE) {
  switch (RF_DIVIDER_value) {
    case 1:
      RF_DIVIDER_value = 0;
      break;
    case 2:
      RF_DIVIDER_value = 1;
      break;
    case 4:
      RF_DIVIDER_value = 2;
      break;
    case 8:
      RF_DIVIDER_value = 3;
      break;
    case 16:
      RF_DIVIDER_value = 4;
      break;
    case 32:
      RF_DIVIDER_value = 5;
      break;
    case 64:
      RF_DIVIDER_value = 6;
      break;
    case 128:
      RF_DIVIDER_value = 7;
      break;
  }
  MAX2870_R[0x02] = BitFieldManipulation.WriteBF_dword(14, 10, MAX2870_R[0x02], R_divider);
  MAX2870_R[0x00] = BitFieldManipulation.WriteBF_dword(15, 16, MAX2870_R[0x00], INT_value);
  MAX2870_R[0x01] = BitFieldManipulation.WriteBF_dword(3, 12, MAX2870_R[0x01], MOD_value);
  MAX2870_R[0x00] = BitFieldManipulation.WriteBF_dword(3, 12, MAX2870_R[0x00], FRAC_value);
  MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(20, 3, MAX2870_R[0x04], RF_DIVIDER_value);
  if (FRACTIONAL_MODE == false) {
    MAX2870_R[0x00] = BitFieldManipulation.WriteBF_dword(31, 1, MAX2870_R[0x00], 1); // integer-n mode
    MAX2870_R[0x01] = BitFieldManipulation.WriteBF_dword(29, 2, MAX2870_R[0x01], 0); // Charge Pump Linearity
    MAX2870_R[0x01] = BitFieldManipulation.WriteBF_dword(31, 1, MAX2870_R[0x01], 1); // Charge Pump Output Clamp
    MAX2870_R[0x02] = BitFieldManipulation.WriteBF_dword(8, 1, MAX2870_R[0x02], 1); // Lock Detect Function, int-n mode
    MAX2870_R[0x05] = BitFieldManipulation.WriteBF_dword(24, 1, MAX2870_R[0x05], 1); // integer-n mode
  }
  else {
    MAX2870_R[0x00] = BitFieldManipulation.WriteBF_dword(31, 1, MAX2870_R[0x00], 0); // frac-n mode
    MAX2870_R[0x01] = BitFieldManipulation.WriteBF_dword(29, 2, MAX2870_R[0x01], 1); // Charge Pump Linearity
    MAX2870_R[0x01] = BitFieldManipulation.WriteBF_dword(31, 1, MAX2870_R[0x01], 0); // Charge Pump Output Clamp
    MAX2870_R[0x02] = BitFieldManipulation.WriteBF_dword(8, 1, MAX2870_R[0x02], 0); // Lock Detect Function, frac-n mode
    MAX2870_R[0x05] = BitFieldManipulation.WriteBF_dword(24, 1, MAX2870_R[0x05], 0); // frac-n mode
  }
```

```
int MAX2870::setPDpolarity(uint8_t PDpolarity) {
  if (PDpolarity == MAX2870_LOOP_TYPE_INVERTING || PDpolarity == MAX2870_LOOP_TYPE_NONINVERTING) {
    MAX2870_R[0x02] = BitFieldManipulation.WriteBF_dword(6, 1, MAX2870_R[0x02], PDpolarity);
    WriteRegs();
    return MAX2870_ERROR_NONE;
  }
  else {
    return MAX2870_ERROR_POLARITY_INVALID;
  }
}
```

MAX2870, SHDN = 1 (register 2, bit 5) olarak ayarlanarak veya CE pini lojik düşük seviyeye ayarlanarak düşük güç moduna alınabilir. Düşük güç modunda, SPI hariç tüm bloklar kapalıdır.

Faz dedektörü polaritesi, etkin bir tersleyici döngü filtresi topolojisi kullanılıyorsa değiştirilebilir. Tersleyici olmayan döngü filtreleri için, PDP = 1 (register 2, bit 6) olarak ayarlanmalıdır. Tersleyici döngü filtreleri için ise, PDP = 0 olarak ayarlanmalıdır.

6	PDP	Phase Detector Polarity	Sets phase detector polarity. 0 = Negative (for use with inverting active loop filters) 1 = Positive (for use with passive loop filers and noninverting active loop filters)

```
int MAX2870::setPowerLevel(uint8_t PowerLevel) {
  if (PowerLevel < 0 && PowerLevel > 4) return MAX2870_ERROR_POWER_LEVEL;
  if (PowerLevel == 0) {
    MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(5, 1, MAX2870_R[0x04], 0);
  }
  else {
    PowerLevel--;
    MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(5, 1, MAX2870_R[0x04], 1);
    MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(3, 2, MAX2870_R[0x04], PowerLevel);
  }
  WriteRegs();
  return MAX2870_ERROR_NONE;
}

int MAX2870::setAuxPowerLevel(uint8_t PowerLevel) {
  if (PowerLevel < 0 && PowerLevel > 4) return MAX2870_ERROR_POWER_LEVEL;
  if (PowerLevel == 0) {
    MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(8, 1, MAX2870_R[0x04], 0);
  }
  else {
    PowerLevel--;
    MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(6, 2, MAX2870_R[0x04], PowerLevel);
    MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(8, 1, MAX2870_R[0x04], 1);
  }
  WriteRegs();
  return MAX2870_ERROR_NONE;
}
```

To enable RFOUT, Register 4, Address 0X04, set bit 4

and 8 to 0.

Register programming order should be address 0x05,

0x04, 0x03, 0x02, 0x01, and 0x00. Several bits are

double-buffered to update the settings at the same time.

See the register descriptions for double-buffered settings

Karşılaştırma ADF4351/MAX2870

github.com/brycecherry75/ADF4351/blob/main/src/ADF4351.cpp

Files | main | ADF4351 / src / ADF4351.cpp | Top

Code | Blame | Raw

```
void ADF4351::setfDirect(uint16_t R_divider, uint16_t INT_value, uint16_t MOD_value, uint16_t
  switch (RF_DIVIDER_value) {
    case 1:
      RF_DIVIDER_value = 0;
      break;
    case 2:
      RF_DIVIDER_value = 1;
      break;
    case 4:
      RF_DIVIDER_value = 2;
      break;
    case 8:
      RF_DIVIDER_value = 3;
      break;
    case 16:
      RF_DIVIDER_value = 4;
      break;
    case 32:
      RF_DIVIDER_value = 5;
      break;
    case 64:
      RF_DIVIDER_value = 6;
      break;
  }
  ADF4351_R[0x02] = BitFieldManipulation.WriteBF_dword(14, 10, ADF4351_R[0x02], R_divider);
  ADF4351_R[0x00] = BitFieldManipulation.WriteBF_dword(15, 16, ADF4351_R[0x00], INT_value);
  ADF4351_R[0x01] = BitFieldManipulation.WriteBF_dword(3, 12, ADF4351_R[0x01], MOD_value);
  ADF4351_R[0x00] = BitFieldManipulation.WriteBF_dword(3, 12, ADF4351_R[0x00], FRAC_value);
  ADF4351_R[0x04] = BitFieldManipulation.WriteBF_dword(20, 3, ADF4351_R[0x04], RF_DIVIDER_value
```

MAX2870.cpp 2

src > MAX2870.cpp > setCPcurrent(float)

```
void MAX2870::setfDirect(uint16_t R_divider, uint16_t INT_value, uint16_t
  switch (RF_DIVIDER_value) {
    case 1:
      RF_DIVIDER_value = 0;
      break;
    case 2:
      RF_DIVIDER_value = 1;
      break;
    case 4:
      RF_DIVIDER_value = 2;
      break;
    case 8:
      RF_DIVIDER_value = 3;
      break;
    case 16:
      RF_DIVIDER_value = 4;
      break;
    case 32:
      RF_DIVIDER_value = 5;
      break;
    case 64:
      RF_DIVIDER_value = 6;
      break;
    case 128:
      RF_DIVIDER_value = 7;
      break;
  }
  MAX2870_R[0x02] = BitFieldManipulation.WriteBF_dword(14, 10, MAX2870_R[0
  MAX2870_R[0x00] = BitFieldManipulation.WriteBF_dword(15, 16, MAX2870_R[0
  MAX2870_R[0x01] = BitFieldManipulation.WriteBF_dword(3, 12, MAX2870_R[0x
  MAX2870_R[0x00] = BitFieldManipulation.WriteBF_dword(3, 12, MAX2870_R[0x
  MAX2870_R[0x04] = BitFieldManipulation.WriteBF_dword(20, 3, MAX2870_R[0x
```

SORUNLAR 2 | ÇIKIŞ | TERMINAL | Python: MAX2870pf

```
pe' and 'int'
PS C:\Users\yusuf\Desktop\MAX2870-main>
```

3 seçim (2 karakter seçildi) Boşluklar: 2 UTF-8 LF {} C++ Win32

github.com/brycecherry75/ADF4351/blob/main/src/ADF4351.cpp

Files | main | ADF4351 / src / ADF4351.cpp | Top

Code | Blame | Raw

```
  ADF4351_ChanStep = value;
  return ADF4351_ERROR_NONE;
}

int ADF4351::setf(char *freq, uint8_t PowerLevel, uint8_t AuxPowerLevel, uint8_t AuxFrequencyDi
  ADF4351_FrequencyError = 0;
  //  calculate settings from freq
  if (PowerLevel < 0 || PowerLevel > 4) return ADF4351_ERROR_POWER_LEVEL;
  if (AuxPowerLevel < 0 || AuxPowerLevel > 4) return ADF4351_ERROR_AUX_POWER_LEVEL;
  if (AuxFrequencyDivider != ADF4351_AUX_DIVIDED && AuxFrequencyDivider != ADF4351_AUX_FUNDAMEN
  if (ReadPFDfreq() == 0) return ADF4351_ERROR_ZERO_PFD_FREQUENCY;

  uint32_t ReferenceFrequency = ADF4351_reffreq;
  ReferenceFrequency /= ReadR();
  if (PrecisionFrequency == false && ADF4351_ChanStep > 1 && (ReferenceFrequency % ADF4351_Char
    return ADF4351_ERROR_PFD_AND_STEP_FREQUENCY_HAS_REMAINDER;
  }

  BigNumber::begin(12); // for a maximum 90 MHz PFD and a 64 RF divider with frequency steps nc

  if (BigNumber(freq) > BigNumber("4400000000") || BigNumber(freq) < BigNumber("34375000")) {
    BigNumber::finish();
    return ADF4351_ERROR_RF_FREQUENCY;
  }

  uint8_t FrequencyPointer = 0;
  while (true) { // null out any decimal places below 1 Hz increments to avoid GCD calculation
```

MAX2870.cpp 2

src > MAX2870.cpp > setCPcurrent(float)

```
  int MAX2870::SetStepFreq(uint32_t value) {
  }
  MAX2870_ChanStep = value;
  return MAX2870_ERROR_NONE;
}

int  MAX2870::setf(char *freq, uint8_t PowerLevel, uint8_t AuxPowerLevel,
  //  calculate settings from freq
  if (PowerLevel < 0 || PowerLevel > 4) return MAX2870_ERROR_POWER_LEVEL;
  if (AuxPowerLevel < 0 || AuxPowerLevel > 4) return MAX2870_ERROR_AUX_POW
  if (AuxFrequencyDivider != MAX2870_AUX_DIVIDED && AuxFrequencyDivider !=
  if (ReadPFDfreq() == 0) return MAX2870_ERROR_ZERO_PFD_FREQUENCY;

  uint32_t ReferenceFrequency = MAX2870_reffreq;
  ReferenceFrequency /= ReadR();
  if (PrecisionFrequency == false && MAX2870_ChanStep > 1 && (ReferenceFre
    return MAX2870_ERROR_PFD_AND_STEP_FREQUENCY_HAS_REMAINDER;
  }

  BigNumber::begin(12); // for a maximum 105 MHz PFD and a 128 RF divider

  if (BigNumber(freq) > BigNumber("6000000000") || BigNumber(freq) < BigNu
    BigNumber::finish();
    return MAX2870_ERROR_RF_FREQUENCY;
  }

  uint8_t FrequencyPointer = 0;
  while (true) { // null out any decimal places below 1 Hz increments to a
    if (freq[FrequencyPointer] == '.') { // change the decimal point to a
      freq[FrequencyPointer] = 0x00;
      break;
    }
```

SORUNLAR 2 | ÇIKIŞ | TERMINAL | Python: MAX2870pf

```
pe' and 'int'
```

APPLICATIONS

Wireless infrastructure (W-CDMA, TD-SCDMA, WiMAX, GSM, PCS, DCS, DECT)
Test equipment
Wireless LANs, CATV equipment
Clock generation

FUNCTIONAL BLOCK DIAGRAM

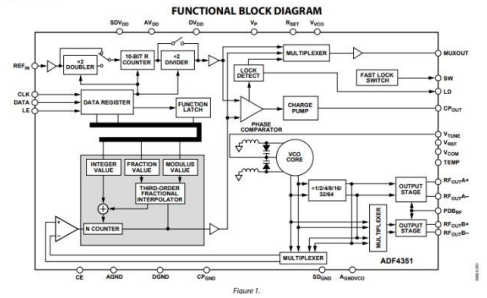


Figure 1.

muted by either software or hardware control.
The MAX2870 is controlled by a 4-wire serial interface and is compatible with 1.8V control logic. The device is available in a lead-free, RoHS-compliant, 5mm x 5mm, 32-pin TQFN package, and operates over an extended -40°C to +85°C temperature range.

- Dual Programmable Outputs
  - -4dBm to +5dBm
- Analog and Digital Lock Detect Indicators
- Hardware and Software Shutdown Control
- Compatible with 1.8V Control Logic

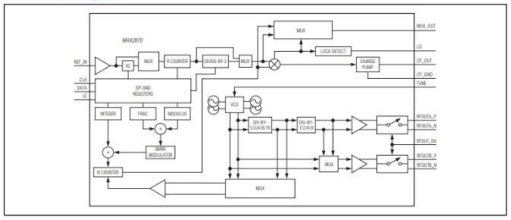
Applications

Wireless Infrastructure
Test and Measurement
Satellite Communications
Wireless LANs/CATV
PMAR/LMAR/Public Safety Radio
Clock Generation

Ordering Information **appears at end of data sheet.**

Typical Application Circuit **appears at end of data sheet.**

Functional Diagram



```
void MAX2870::ReadCurrentFrequency(char *freq)
```

Bu fonksiyon, MAX2870 frekans sentezleyici çipinden mevcut frekans bilgisini okur ve bu bilgiyi belirtilen karakter dizisine (**`freq`**) yazarak döndürür.

```
void MAX2870::init(uint8_t SSpin, uint8_t LockPinNumber, bool Lock_Pin_Used, uint8_t CEpinNumber, bool CE_Pin_Used)
```

Bu fonksiyon, Entegrenin inital parametrelerini set eder. PinMode, pin seçimi gibi parametreleri belirler.

```
int MAX2870::SetStepFreq(uint32_t value)
```

Adım frekansını (step frequency) ayarlar.

```
int  MAX2870::setf(char *freq, uint8_t PowerLevel, uint8_t AuxPowerLevel, uint8_t AuxFrequencyDivider, bool PrecisionFrequency, uint32_t MaximumFrequencyError, uint32_t CalculationTimeout)
```

(304 Satırlık büyük fonksiyon)

Genel Olarak;

1. Kullanıcı tarafından belirtilen frekans değerinin geçerliliğini kontrol eder. Frekans değeri, belirli bir aralıkta olmalıdır. Ayrıca, frekans değeri **BigNumber** kütüphanesi kullanılarak büyük sayılar olarak işlenir.
2. Belirli bir adım frekansı (step frequency) kullanılıyorsa ve hassas frekans (Precision Frequency) modu etkinleştirilmişse, adım frekansının belirtilen frekans değerine tam olarak bölünüp bölünemeyeceğini kontrol eder. Eğer bölünemiyorsa, bir hata kodu döndürür.
3. Frekans değeri, çıkış frekansına bölünebilecek şekilde düzeltilir. Bu, istenen frekansın çıkış frekansı modlarına uygun hale getirilmesini sağlar.
4. MAX2870'nin çeşitli parametreleri hesaplanır ve ayarlanır. Bu parametreler arasında bölücü seçimi, çıkış modu (integer veya fractional), kilitleme tespiti hızı gibi değerler bulunur.
5. Hesaplamaların doğruluğunu kontrol etmek için zaman aşımı (timeout) mekanizması kullanılır. Belirli bir süre içinde hesaplama tamamlanamazsa, bir uyarı döndürülür.
6. Elde edilen parametreler, MAX2870'nin kayıtlarına yazılır ve çip yapılandırması tamamlanır.
7. Son olarak, frekans hatası hesaplanır ve belirli bir tolerans içinde olup olmadığı kontrol edilir. Eğer toleransın dışında ise, bir uyarı döndürülür.

```
void MAX2870::setfDirect(uint16_t R_divider, uint16_t
INT_value, uint16_t MOD_value, uint16_t FRAC_value, uint8_t
RF_DIVIDER_value, bool FRACTIONAL_MODE) {
```

Bu fonksiyon, MAX2870 frekans sentezleyici çipinin frekansını doğrudan belirtilen değerlere ayarlar.

```
int MAX2870::setPowerLevel(uint8_t PowerLevel)
```

Bu fonksiyon, MAX2870 frekans sentezleyici çipinin güç seviyesini belirlemek için kullanılır.

```
int MAX2870::setAuxPowerLevel(uint8_t PowerLevel)
```

Bu fonksiyon, MAX2870 frekans sentezleyici çipinin yardımcı (auxiliary) güç seviyesini belirlemek için kullanılır.

```
int MAX2870::setCPcurrent(float Current)
```

Charge pump akımını ayarlamak için kullanılır.

```
int MAX2870::setPDpolarity(uint8_t PDpolarity)
```

Phase detector polaritesini ayarlamak için kullanılır. Faz kilidi, bir frekans sentezleyicinin çıkış frekansını istenen referans frekansıyla karşılaştırır ve herhangi bir faz farkını düzeltmek için geri besleme sağlar.